Responsive
Web Development with HTML5 and CSS
Building Modern and User-Friendly Websites for All Devices
SAMMIE SMITH

# Responsive Web Development with HTML5 and CSS

**Building Modern and User-Friendly Websites for All Devices**

**SAMMIE SMITH**

Copyright © 2023 **Sammie Smith**

**All Rights Reserved**

Without the publisher's prior written consent and except for what is permitted by American copyright law and fair use, no part of this book or any excerpts from it may be duplicated in any way, stored in any database, or transmitted in any way via any mechanism - mechanical, electronic, photocopy, recording, or otherwise.

**Disclaimer and Terms of Use**

The publishers and anyone involved in the production of this book made every effort to ensure its quality. The correctness, application, suitability, or completeness of the contents of this book are not guaranteed by the authors or publishers. This book only includes entirely educational information. As a result, you are accepting complete responsibility for your actions if you choose to put the concepts in this book into practice.

*Printed in the United States of America*

# TABLE OF CONTENTS


**TABLE OF CONTENTS**

**INTRODUCTION**

**CHAPTER ONE**

**GETTING STARTED WITH HTML**

*What is HTML?*

*What is New in HTML5*

*Why Learn HTML5*

*Choosing a Text Editor or IDE*

*Understanding HTML5 Semantic Markup*

*How is HTML used in Web Development?*

*Understanding the basic structure of an HTML document*

*Basic Tags and Attributes*

*Creating an HTML Document*

*Setting up an HTML Document*

*Adding Content to an HTML Document*

*Understanding HTML Tags, Attributes, and Values*

Adding Comments to an HTML Document

**CHAPTER TWO**

**HTML TEXT FORMATTING**

*Understanding HTML Headings*

*Understanding HTML Paragraphs*

*Understanding HTML Lists*

Unordered Lists

Ordered Lists

Nested Lists

*Understanding HTML links*

Relative URLs

Linking to Specific Parts of a Page

*How to Format Text using HTML Tags*

*How to Add Images to an HTML Document*

*How to Add Videos to an HTML Document*

*Using Audio in HTML5*

**CHAPTER THREE**

**UNDERSTANDING FORMS AND INPUT**

*Creating Forms*

*Text Inputs and Labels*

*Checkboxes and Radio Buttons*

*Select Menus and Option Groups*

*Text Areas and Buttons*

*Understanding Form Validation*

**CHAPTER FOUR**

**UNDERSTANDING TABLES AND LISTS**

*Creating Tables*

*Adding Rows and Columns*

*Styling Tables with CSS*

**CHAPTER FIVE**

**ADVANCED HTML5 FEATURES**

*Canvas and SVG graphics*

How to Apply Canva and SVG Graphics

*Understanding Web Storage*

*Understanding Geolocation*

*Understanding Web Workers*

*Understanding Drag and Drop*

*Understanding Web Sockets*

*Accessibility and SEO*

*Making your HTML5 Code Accessible*

*Using Semantic Markup*

*Applying Semantic Markup*

*Optimizing for Search Engines*

**CHAPTER SIX**

**HTML PRACTICAL EXERCISES**

*Exercise 1: Creating a Login Page*

*Exercise 2: Creating a Registration Page*

*Exercise 3: Create a Simple Personal Portfolio Website*

**CHAPTER SEVEN**

**INTRODUCTION TO CSS**

*What is CSS?*

*How is CSS used in Web Development?*

*How CSS works with HTML*

*Understand the Difference Between CSS and HTML*

*Benefits of using CSS*

*Basic CSS Syntax*

*Writing CSS rules*

*Selectors and Declarations*

*Comments in CSS*

*External and Internal CSS*

*Linking an External CSS*

**CHAPTER EIGHT**

**CSS STYLE PROPERTIES**

*Text Properties*

*What are CSS Text Properties?*
- Applying Text Properties
- Font Properties
- Applying Font Properties
- Color Properties
- Applying Color properties

*Background Properties*
- CSS Box Model
- Applying CSS Box Model

*Understanding the Box Model*

*Margin, Border, and Padding*

*Width and Height*

*CSS Layouts*

*CSS Floats*
- Applying Float

*Positioning*
- Applying Positioning

*Display*
- Applying Display

*Understanding CSS Grid*
- Applying CSS Grid

*CSS Animations*
- Applying CSS Animations

*CSS transitions*

*Animating with Keyframes*

*Undesratanding Flexbox*
Applying Flexbox

*CSS Variables*

*Scalable Vector Graphics (SVG)*

*Writing Efficient CSS*

*Debugging CSS*

*The Future of CSS*

**CHAPTER NINE**

**HTML AND CSS PRACTICAL EXERCISES**

*Exercise 1: Design a simple login page with HTML and CSS*

*Exercise 2: Build a simple landing page for a product with HTML and CSS.*

*Exercise 3: Create a responsive pricing table with HTML and CSS*

**CHAPTER TEN**

**RESPONSIVE WEB DESIGN**

*Introduction to Responsive Design*

*Best Practices for Designing Responsive Websites*

*Tools and Technologies used to Create Responsive Website*

*Understanding Media Queries*
Applying Media Queries

*Understanding Mobile-first Design Approach*
Applying Mobile-first Design

*Responsive Frameworks*

*Common Challenges and Solutions Associated with Responsive Web Design*

*Optimizing Images and Managing Complex Layouts*

**CHAPTER ELEVEN**

**RESPONSIVE WEBSITES PRACTICAL EXERCISES**

*Exercise 1: Design a responsive restaurant website*

*Exercise 2: Create a Responsive Photographer Portfolio*

*Exercise 3: Develop a Responsive E-commerce Website*

**Conclusion**

**INDEX**

# INTRODUCTION

With the proliferation of smartphones and tablets, responsive web design has become an essential aspect of modern web development. Responsive web design is a web development approach that aims to provide users with an optimal viewing experience across a wide range of devices, from desktop computers to mobile phones. HTML (Hypertext Markup Language) and CSS (Cascading Style Sheets) are two of the core technologies used in building responsive websites. HTML is used for structuring the content of a web page, while CSS is used for styling and layout. Together, these technologies enable developers to create flexible and adaptable websites that can adjust their layout and content to match the size and resolution of the user's device.

Throughout the book, we will focus on best practices for building responsive websites that are accessible, usable, and performant. We will also cover common pitfalls and mistakes that developers should avoid, as well as tips and tricks for optimizing performance and user experience. Throughout the book, you will work on hands-on exercises and projects, which will help you apply what you have learned and build your skills. By the end of the book, you will have the knowledge and confidence to create your web pages and customize them with CSS.

By the end of this book, you will have a solid understanding of the principles and techniques of HTML and CSS for responsive web development. You will be able to build responsive websites that are

optimized for a variety of devices and screen sizes, and that provide users with a seamless and enjoyable experience. Whether you're a beginner or an experienced web developer, this book will help you take your HTML and CSS skills to the next level. So, let's get started!

CHAPTER ONE

# GETTING STARTED WITH HTML

In this chapter, we will introduce the basics of HTML, including tags, attributes, and elements. We will explain how to create an HTML document, set up a basic structure, and add content.

## What is HTML?

HTML stands for Hypertext Markup Language. It is a standard markup language used to create web pages and other types of digital content that can be displayed on the internet. HTML provides a way to structure and format content such as text, images, and multimedia elements, allowing web browsers to render them in a visually appealing and organized manner.

HTML works by using a set of tags and attributes to define the structure and appearance of web page content. These tags are enclosed in angle brackets (< >) and specify the type of content they contain. For example, the <h1> tag is used to define a heading, while the <p> tag is used to define a paragraph. Attributes are used to provide additional information about a tag, such as its class, id, or style.

Web developers use HTML in conjunction with CSS (Cascading Style Sheets) and JavaScript to create dynamic, interactive web pages. HTML5 is the latest

version of the HTML specification and includes new features such as video and audio elements, as well as improved support for mobile devices.

# What is New in HTML5

HTML5 introduced several new features and improvements over previous versions of HTML. Some of the key new features include:

- **Video and Audio Features:** Audio and video elements are two essential new features of HTML5. It makes it possible for site designers to add a video or audio clip to their pages. HTML5's video element can be styled with CSS and CSS. Changes can be made to borders, opacity, gradients, reflections, transitions, transformations, and even animations. By making the code available for use in embedding their films on other websites, YouTube also announces video embedding. It helps the web become more and more interactive with multimedia. To embed any audio clip into a webpage on the Internet, HTML5 provides a new element called the audio tag that is available for use.
- **Header and Footer:** The need for a div> tag to divide the two portions is gone thanks to these new tags. The header is at the top of the page, while the footer is at the bottom. If you utilize the HTML5 elements header and footer, the browser will understand what to load first and what to load later.
- **New input tag types included**: An outdated attribute called "input" has been given new life in HTML by taking on new values like "email," "month," "number," "range," "search," "tel," "color," "week," "URL," "time," "date," "DateTime-local," and so on. The input tag may now contain the following new values:
    1. **ContentEditable:** It's a feature that enables content modification by the user. It makes it simple to tell if what you see is what you get. You will be able to edit the content by clicking on it.
    2. **Progress**: This tag is used to keep track of a job's progress as it is finished. The JavaScript progress tag

is compatible with it. It has a progress bar-like appearance.

3. **Section**: A document can be divided into sections or components using the section tag. For instance, an article might have several sections, including a header, footer, section for the main content, and the newest news, among others.
4. **Main:** The main tag is used to summarize the page's main content. A document may include just one main tag, and this tag may not be contained within the article aside footer, or header tags. There is no header, footer, or navigation bar.

- **Figure and figcaption:** The addition of a figure and a caption to a document was not before possible. With the advent of the figure and figcaption tags, it is now semantically possible to embed an image together with its explanation into a website. The syntax is as follows:

```
<figure>
    <img src="image/image-1.jpg" alt="About ADMEC" />
        <figcaption>
            <p>This is our institute </p>
        </figcaption>
</figure>
```

- **Placeholders:** Before, we had to utilize a little JavaScript to create placeholders for text fields. Sure, you can first alter the value property as you see fit, but if the user deletes that information and then clicks away, the input will be left empty. Using the placeholder property corrects this.
- **Preload Videos**: It's an excellent feature for sharing videos. It explains how to upload the video and how long it will take the website to load. This notifies the browser about the improvements made to the user experience of the webpage. Although it's not a requirement, you should nonetheless add this feature. It contributes to a more realistic representation of the page.
- **Controlling the display:** The behavior of elements is determined by the display property. If this attribute is not

specified, the default values are applied.

- **Regular Expressions**: To add a certain pattern as an input, we may use a regular expression. For instance, the most common pattern is [A-Za-z] 5,11. Both capital and lowercase letters are acceptable. Additionally, it says that the minimum and maximum character lengths are five and eleven, respectively.
- **Adaptability:** Since its creation, HTML5 has significantly contributed to a website's ability to offer the best accessibility features. As a result, the website's usage has been simplified. The accessibility features of HTML5 allow users with virtually any type of disability, including vision impairment, color blindness, impaired vision, blindness, and more, to access websites. For instance, the best illustration of providing accessibility in forms is validation. Labels must be seen clearly.
- **Inline components**: These inline items are very helpful in keeping code current:
    1. **mark:** It highlights items that have been in some way marked.
    2. **Time**: The website's current time and date are shown using this.
    3. **Meter**: It shows how much space is left available on the storage disk.
    4. **Progress bar**: The progress bar lets you monitor the status of a task you have been given.
- **Support for Dynamic Pages:** Nowadays, dynamic and interactive websites are preferred over static ones. The website has a dynamic vibe because of a few factors:
    1. **Mark**: It highlights items that have been in some way marked.
    2. **Time**: The website's current time and date are shown using this.
    3. **Meter**: It shows how much space is left available on the storage disk.
    4. **The progress:** The progress bar lets you monitor the status of a task you have been given. Other factors also contribute to the website's dynamic nature.
- **Email as a property:** The browser automatically takes the instruction from the code to produce an email in the correct and valid format when we specify the type of email in a form. Earlier browsers did not make this possible.
- **Cryptographic Nonces**: In this most recent version of HTML, we may now apply cryptographic nonces to all

styles and scripts. Our standard practice is to use the nonce attribute inside the script and style elements. In essence, this nonce tag generates a random integer that is only used once. As a result, the page is updated each time it is refreshed. It's a great feature since it can be used to increase the content of the page's security, allowing the website to declare and choose a certain script or style.

- **Reverse Links:** The rev attribute for reverse links is back in use with HTML 5.1. In essence, it gives web users the ability to use the link and anchor tag elements once more. It also explains the relationship between the linked document and the current document in reverse.
- **Photographs with a width of Zero**: Now, website developers can adjust the width of the images to zero. When there is no need to show them to customers, such as when monitoring picture files because they would otherwise take up more space, this capability is helpful. Utilizing images with a zero width and a blank alt tag is advised.
- **Canvas in HTML5:** A canvas is a rectangular area that can be used for pixel-level actions such, among other things, drawing a line, box, or circle, or running images. The support for canvas regions in HTML5 has been added. Below is a sample of a program's code.

```
<canvas id="myCanvas" width="200" height="100">
    Fallback content, when canvas is not supported by the browser.
    </canvas>
```

# Why Learn HTML5

There are many reasons why learning HTML5 is a valuable skill for anyone interested in web development or design. Here are a few of the main reasons:

- **Essential for building websites**: HTML5 is the foundation of all modern websites. Understanding HTML5 is essential

for anyone who wants to create, edit, or customize websites.

- **Improved web standards**: HTML5 is designed to improve web standards and make the web more accessible and user-friendly. By learning HTML5, you can help ensure that your websites are built to these standards.
- **Better multimedia support**: HTML5 introduces new elements and features for embedding multimedia content, such as videos, audio, and images. This makes it easier to create rich and engaging websites.
- C**ross-platform compatibility**: HTML5 is supported across all major web browsers and devices, including desktops, laptops, tablets, and smartphones. This makes it easier to create websites that work seamlessly across different platforms and devices.
- **Web application development**: HTML5 also provides features and APIs for building advanced web applications, such as local storage, geolocation, web sockets, and more. These features can help you create more interactive and engaging web applications.
- **High demand for HTML5 developers**: There is a high demand for web developers who are proficient in HTML5, especially those who also have experience with other web development technologies like CSS and JavaScript. Learning HTML5 can open up many job opportunities in the tech industry.

# Choosing a Text Editor or IDE

Choosing a text editor or Integrated Development Environment (IDE) is an important step when setting up your HTML5 development environment. Here are some factors to consider when choosing a text editor or IDE:

- **Features**: Look for a text editor or IDE that has features that will make your development process easier, such as syntax highlighting, auto-completion, code folding, and code snippets.

- **Ease of use**: Choose a text editor or IDE that is easy to use and has an intuitive user interface. The text editor or IDE should be easy to navigate and allow you to quickly find the tools you need.
- **Compatibility**: Make sure the text editor or IDE you choose is compatible with your operating system and other tools you plan to use, such as your web browser and web server.
- **Community support**: Choose a text editor or IDE that has a large and active community of users, as this can be a valuable resource for getting help, finding plugins and extensions, and sharing your work.
- **Price**: Consider the cost of the text editor or IDE. Some text editors, like Sublime Text and Atom, offer free versions with limited features, while others, like Visual Studio Code and WebStorm, require a paid license for full access to all features.

Here are some popular options for text editors and IDEs for HTML5 development:

- **Visual Studio Code**: A free and open-source IDE developed by Microsoft, with support for HTML, CSS, and JavaScript.
- **Sublime Text:** A lightweight text editor with a free version and a paid license for additional features.
- **Atom**: A free and open-source text editor developed by GitHub, with support for HTML, CSS, and JavaScript.
- **Brackets**: A free and open-source text editor developed by Adobe, with support for HTML, CSS, and JavaScript.
- **WebStorm**: A paid IDE developed by JetBrains, with support for HTML, CSS, and JavaScript.

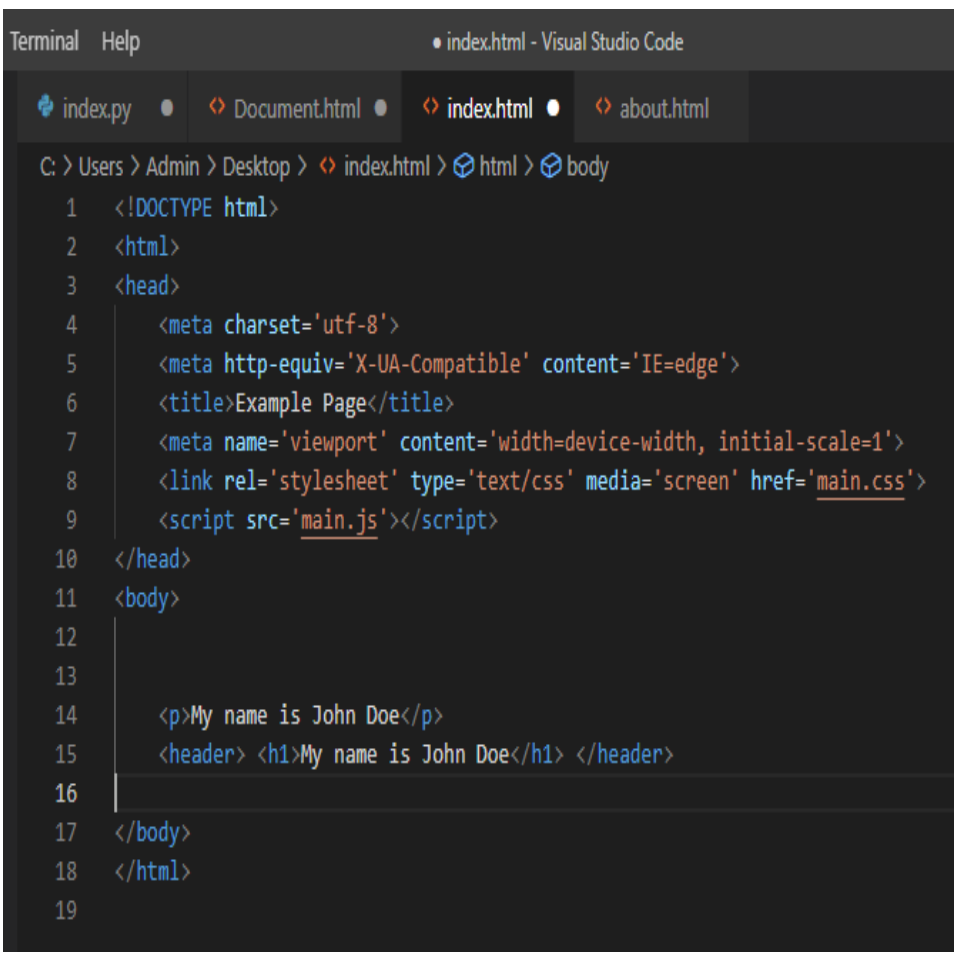
Ultimately, the choice of a text editor or IDE is a personal preference, and it's important to find the one that works best for you and your workflow. Every exercise contained in this book is deployed using Visual Studio Code.

# Understanding HTML5

# Semantic Markup

Semantic markup is the practice of using HTML tags that accurately describe the meaning and purpose of the content they contain. In other words, semantic markup is the use of HTML tags to give meaning to the content of a web page, rather than just for visual presentation.

For example, consider the following two HTML code snippets:



In the first example, the content is simply marked up as a paragraph. In the second example, however, the content is marked up using a header and a heading element, which gives it more meaning and context.

By using semantic markup, web developers can make their web pages more accessible to all users, including those who use assistive technologies, such as screen readers. Semantic markup also helps search engines better understand the content of a web page, which can improve search engine rankings.

Some examples of semantic HTML tags include:

- **<header>**: Used to define the header of a section or web page
- **<nav>**: Used to define a section of navigation links
- **<article>**: Used to define a standalone piece of content, such as a blog post or news article
- **<section>**: Used to define a section of related content
- **<footer>**: Used to define the footer of a section or web page
- **<aside>**: Used to define content that is related to the surrounding content, but not a part of it

Using semantic markup can help make your HTML code more meaningful, readable, and accessible.

# How is HTML used in Web Development?

HTML is a fundamental part of web development and is used to create the structure and content of web pages. When a user visits a website, their web browser retrieves the HTML code from the server and

interprets it to display the content on the screen.

HTML is used in web development in the following ways:

- **Creating the structure of web pages**: HTML provides the basic structure of a web page by defining headings, paragraphs, lists, tables, and other structural elements.
- **Adding content to web pages**: HTML is used to add text, images, videos, audio files, and other types of content to web pages.
- **Creating links**: HTML allows developers to create links between web pages, enabling users to navigate between them.
- **Building forms**: HTML is used to create web forms that allow users to submit data to a website, such as contact information or user login details.
- **Adding metadata**: HTML allows developers to add metadata to web pages, such as title tags, meta descriptions, and keywords. This information is used by search engines to index and rank web pages.

Note that HTML is a critical component of web development, and knowledge of HTML is essential for creating functional and attractive web pages.

# Understanding the basic structure of an HTML document

here's a brief overview of the basic structure of an HTML document:

- **<!DOCTYPE html>**: This is the document type declaration, which tells the web browser which version of HTML the document is written in. For HTML5, this should always be the first line of your document.

- **<html>**: The **<html>** tag is used to define the beginning and end of the HTML document. All the content of the document should be enclosed within these tags.
- **<head>**: The **<head>** tag contains meta-information about the document such as the title of the page, links to CSS stylesheets, and other metadata that is not displayed in the browser.
- **<title>**: The **<title>** tag is used to define the title of the document. This title appears in the browser's title bar and is also used by search engines to describe the page.
- **<body>**: The **<body>** tag is used to define the main content of the document. This is where you put all of the visible content that will be displayed in the browser.
- **<h1>**, **<h2>**, **<h3>**, **<h4>**, **<h5>**, **<h6>**: These are the heading tags, which are used to define headings and subheadings within the document. **<h1>** is the largest heading and **<h6>** is the smallest.
- **<p>**: The **<p>** tag is used to define paragraphs of text.
- **<a>**: The **<a>** tag is used to create hyperlinks to other web pages or other sections within the same document.
- **<img>**: The **<img>** tag is used to insert images into the document.
- **<ul>** and **<li>**: These are the unordered list and list item tags, respectively. They are used to create bulleted lists.
- **<ol>** and **<li>**: These are the ordered list and list item tags, respectively. They are used to create numbered lists.
- ** **: The ** ** tag is used to group related elements in the document. It is often used in conjunction with CSS to apply styles to a group of elements.

That's a quick overview of the basic structure of an HTML document. Keep in mind that there are many more tags and attributes available in HTML, but these are some of the most commonly used ones.

# Basic Tags and Attributes

HTML5 includes a wide variety of tags and attributes that you can use to create content and structure web pages. Here are some of the most common HTML

tags and attributes:

**Tags:**

- **<html>**: The root tag that wraps around all other content on the web page.
- **<head>**: Contains meta-information about the web page, such as the page title and links to external stylesheets and scripts.
- **<body>**: Contains the main content of the web page.
- **<h1>** to **<h6>**: Used to create headings of varying levels of importance.
- **<p>**: Used to create paragraphs of text.
- **<a>**: Used to create hyperlinks to other web pages or resources.
- **<img>**: Used to insert images into the web page.
- **<ul>** and **<ol>**: Used to create unordered and ordered lists, respectively.
- **<li>**: Used to create list items.
- ** ** and ****: Used to group content and apply styles to it.
- **<form>**: Used to create a form for user input.

## Attributes:

- **id**: Used to uniquely identify an element on the web page.
- **class**: Used to group elements together and apply styles to them.
- **src**: Used to specify the URL of an image or other resource.
- **alt**: Used to provide alternative text for images, which is used by screen readers and search engines.
- **href**: Used to specify the URL that a hyperlink points to.
- **target**: Used to specify where the linked resource should be displayed (e.g. in a new window or tab).
- **type**: Used to specify the type of input in a form field (e.g. text, password, checkbox, radio button).

These are just a few of the many HTML tags and attributes available. As you become more familiar with HTML, you'll discover additional tags and attributes that can be used to create more complex and

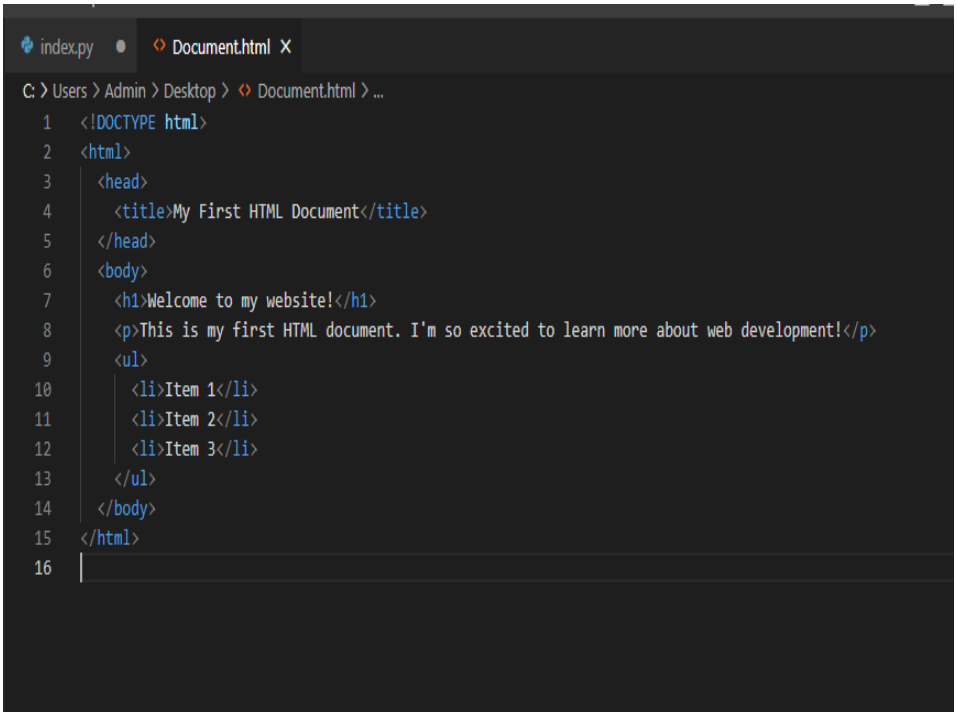
interactive web pages.

# Creating an HTML Document

Here are the basic steps to create an HTML document:

- Open a text editor on your computer like Visual studio.
- Start with the HTML doctype declaration, which is the first line of an HTML document. It looks like this: **<!DOCTYPE html>**
- Next, create the HTML document structure by adding the **<html>** element. This element contains the entire HTML document and has two parts: the **<head>** section and the **<body>** section.
- Inside the **<head>** section, you can add the title of your document using the **<title>** element. This is the text that appears in the browser's title bar.
- Inside the **<body>** section, you can add the content of your document using various HTML elements such as **<p>** for paragraphs, **<h1>** for headings, **<ul>** and **<li>** for lists, etc.
- Once you've added all the content you want, save your file with a .html extension.

Here's an example of what your HTML document might look like:



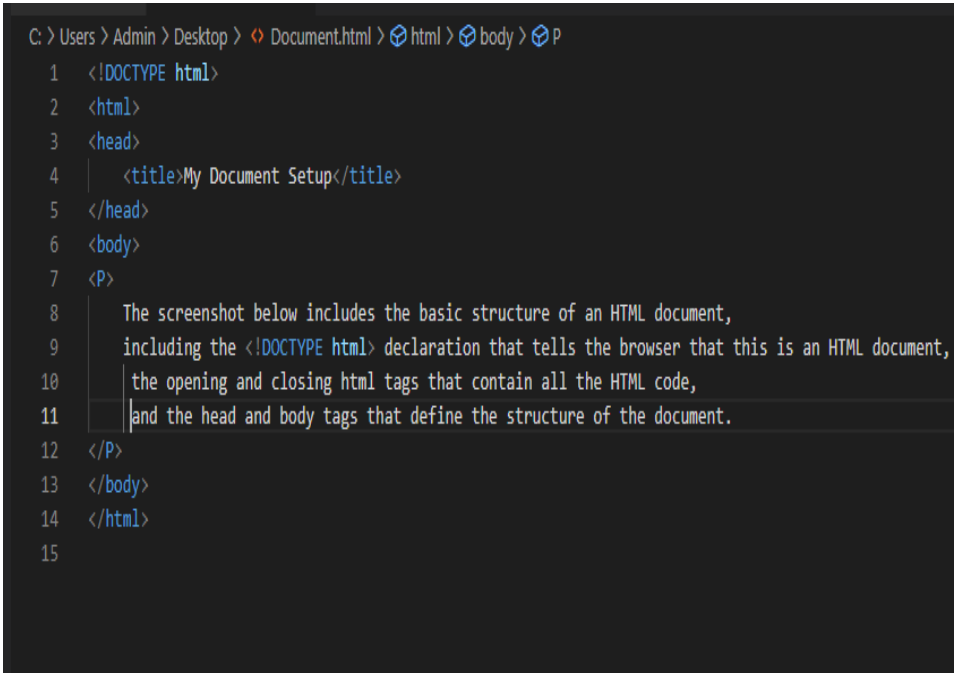

This is just a basic example, but you can add more complex elements, styles, and functionality as you become more comfortable with HTML.

# Setting up an HTML Document

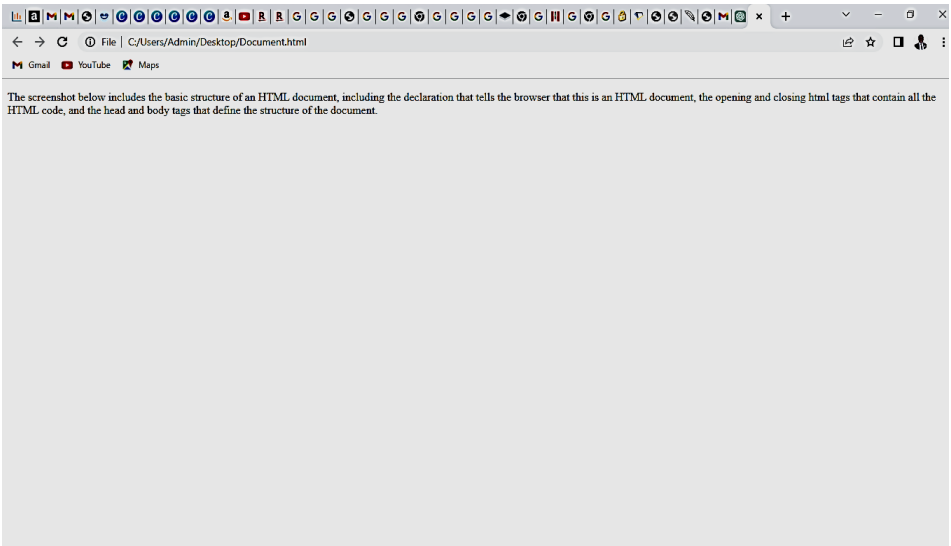
The screenshot below includes the basic structure of an HTML document, including the **<!DOCTYPE html>** declaration that tells the browser that this is an HTML document, the opening and closing **html** tags that contain all the HTML code, and the **head** and **body** tags that define the structure of the document.

To set up an HTML document, you can follow these steps:

- Open a text editor such as Notepad, Sublime Text, or Visual Studio Code.
- Type in the following code to start your HTML document:



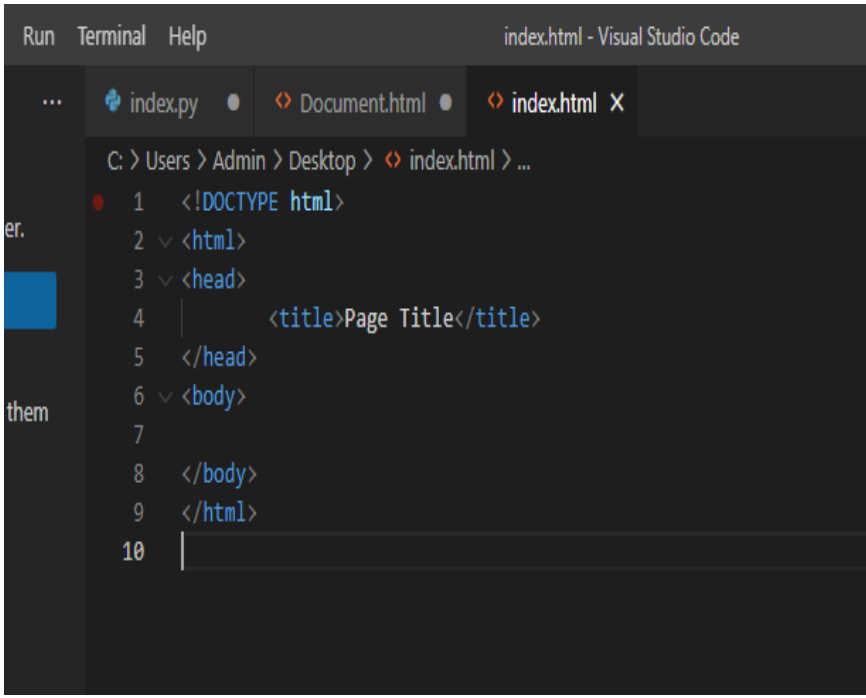

- Replace the **Page Title** with the title of your page. This title will appear in the browser's tab or title bar.
- Add any content you want to include within the **body** tags. This can include headings, paragraphs, images, links, and other HTML elements.
- Save the file with a **.html** extension, such as **index.html**.
- Open the file in a web browser to view your HTML document. See the screenshot below.


File | C:/Users/Admin/Desktop/Document.html
Gmail
YouTube
Maps
The screenshot below includes the basic structure of an HTML document, including the declaration that tells the browser that this is an HTML document, the opening and closing html tags that contain all the HTML code, and the head and body tags that define the structure of the document.

# Adding Content to an HTML Document

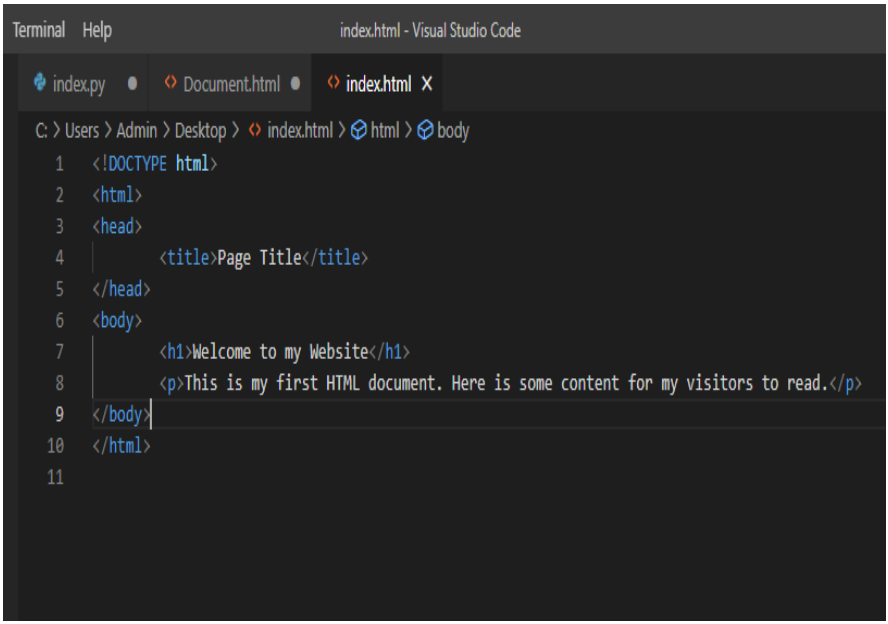
To add content to an HTML document, you can use HTML tags to structure and format the content. Here are the basic steps:

- Open a text editor, such as Notepad or Sublime Text.
- Create a new file and save it with a .html extension, for example, "index.html".
- Begin by adding the basic structure of an HTML document using the following tags:

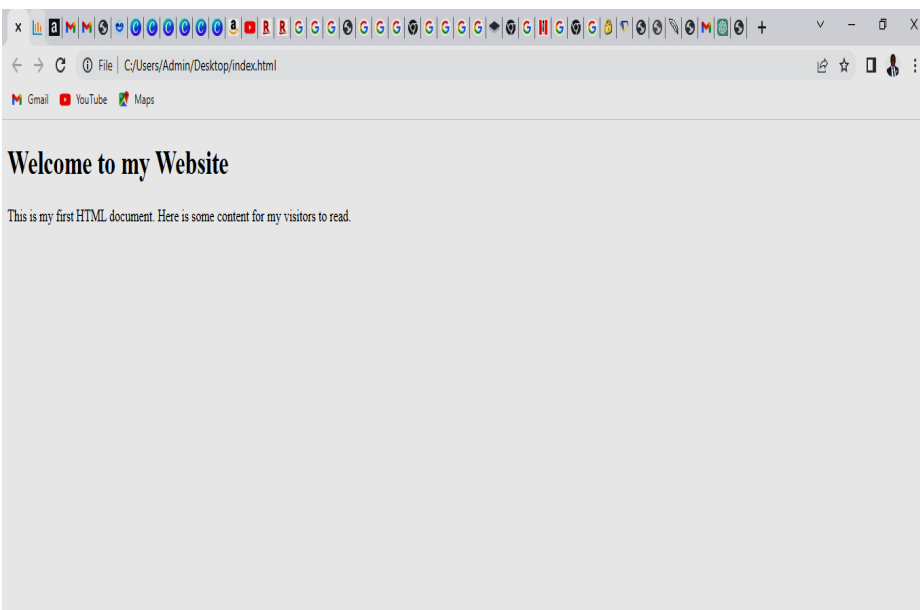



- The **<!DOCTYPE html>** tag specifies the document type and should be included at the beginning of every HTML document.
- The **<html>** tag represents the root element of an HTML document.
- The **<head>** tag contains metadata about the document, such as the title of the page, which is displayed in the browser tab.
- The **<title>** tag specifies the title of the page.
- The **<body>** tag contains the visible content of the document.
- You can add content to the **<body>** tag using various HTML tags such as:
  1. **<h1>** to **<h6>** for headings of different sizes
  2. **<p>** for paragraphs
  3. **<ul>** and **<ol>** for lists
  4. **<a>** for links
  5. **<img>** for images
  6. **<table>** for tables
  7. ** ** and **** for grouping and styling content

- For example, to add a heading and paragraph to your HTML document, you can use the following code:




- Save the file and open it in a web browser to see the rendered content.


File C:/Users/Admin/Desktop/index.html
Gmail
YouTube
Maps
Welcome to my Website
This is my first HTML document. Here is some content for my visitors to read.

# Understanding HTML Tags, Attributes, and Values

HTML (Hypertext Markup Language) is a markup language used to create web pages. It consists of tags, attributes, and values, which are used to structure and format content on a web page.

HTML Tags: HTML tags are used to define the structure of the content on a web page. They are enclosed in angle brackets (< >) and come in pairs, with a start tag and an end tag. The start tag begins with the name of the tag, and the end tag begins with a forward slash (/) followed by the tag name. For example:

```
<p>This is my first HTML document. Here is some content for my visitors to read.</p>
```

In the above example, the ** ** tag is the start tag and the ** ** tag is the end tag. The content "This is a paragraph." is enclosed between these tags and is formatted as a paragraph.

HTML Attributes: HTML attributes are used to provide additional information about HTML elements. They are added to the start tag of an HTML element and consist of a name and a value. The name and value are separated by an equal sign (=), and the value is enclosed in quotation marks. For example:

```
<img src="image.jpg" alt="An image">
```

In the above example, the **src** and **alt** attributes are added to the **<img>** tag. The **src** attribute specifies the URL of the image file, and the **alt** attribute provides alternative text for the image.

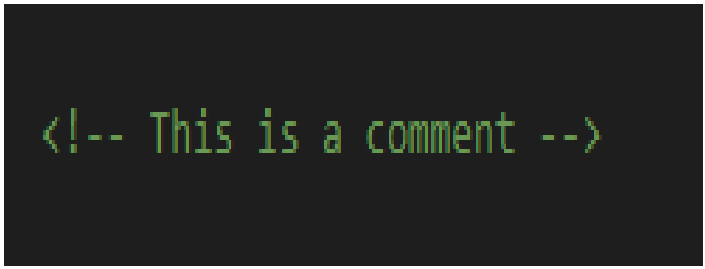
HTML Values: HTML values are used to provide a value for an attribute. They are enclosed in quotation marks and can be either a single word or a sentence. For example:

```
<a href="https://www.google.com">Visit Example</a>
```

In the above example, the **href** attribute is used to provide the URL of the page that the link should point to. The value of the **href** attribute is "https:// www.google.com". The text "Visit Example" between the **<a>** and **</a>** tags is the content of the link.

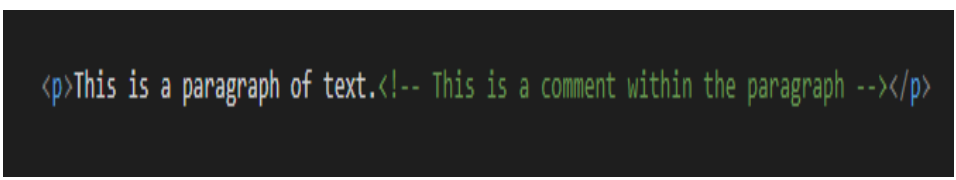
## Adding Comments to an HTML Document

Comments in an HTML document are used to add notes or explanations that are not visible to the user but can be read by developers who are reviewing the code. To add comments to an HTML document, you can use the following syntax:




Anything that is written between the **<!--** and **-->** tags will be considered a comment and will not be rendered by the browser. You can add comments anywhere in the HTML code, including within tags, after tags, or between content.

Here's an example of adding a comment within an HTML tag:




In this example, the text "This is a comment within the paragraph" is a comment and will not be visible to the user.

CHAPTER TWO

# HTML TEXT FORMATTING

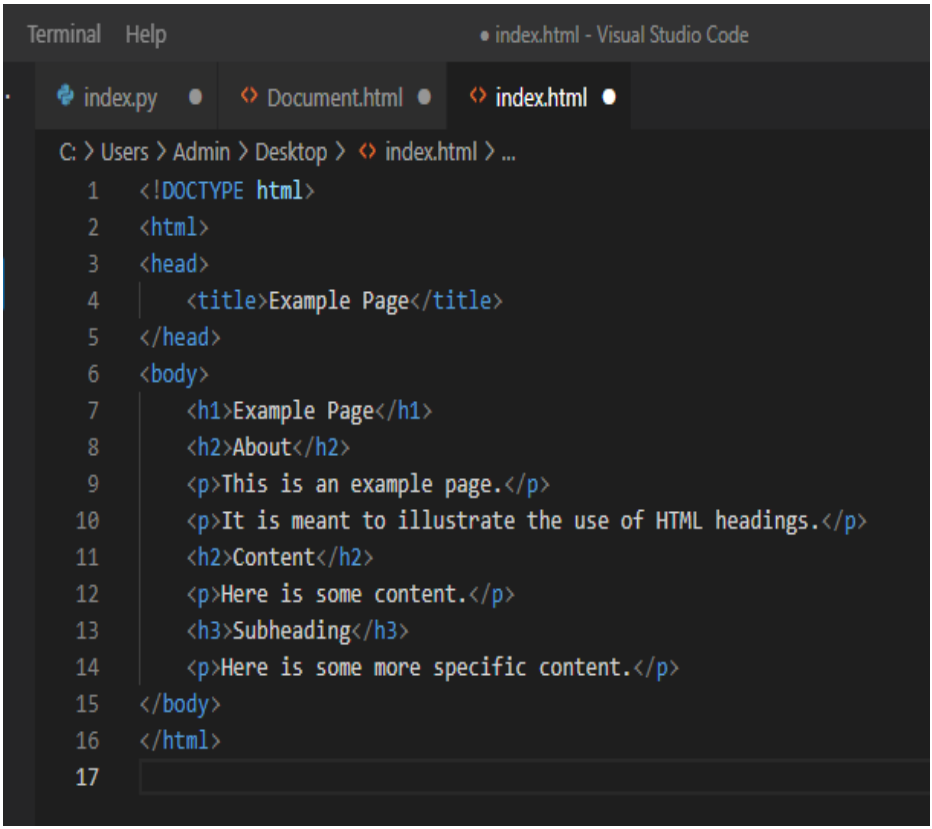
In this chapter, we will explore text formatting in HTML. We will cover headings, paragraphs, lists, and links. We will explain how to format text using HTML tags and how to add images to an HTML document. In HTML, text formatting is essential to make your content look presentable and readable.

## Understanding HTML Headings

HTML headings are used to define the headings or titles of a web page or a section within a web page. There are six levels of headings in HTML, which are represented by the h1 to h6 tags.

The h1 tag is the highest level of heading and is usually used for the main title of the page. The h2 tag is used for subheadings, and so on, with each subsequent level of heading being used for increasingly smaller and more specific sections of the page.

Here is an example of how HTML headings might be used on a simple web page:




In the example above, the h1 tag is used for the main title of the page, the h2 tag is used for two subheadings, and the h3 tag is used for a subheading under the second subheading. The actual text of the headings and content can be customized as needed to suit the needs of the page.

# Understanding HTML Paragraphs

HTML paragraphs are used to structure text into paragraphs, which are blocks of text that typically convey a complete idea or thought.

To create a paragraph in HTML, you can use the  element. The  element is a container tag that is used to enclose text content to create a paragraph. Here's an example of how to create a paragraph in HTML:

* This is a paragraph. *

In this example, the text "This is a paragraph." is enclosed within the  tags, indicating that it is a paragraph.

You can also add additional attributes to the  element to provide more information about the paragraph, such as the class or ID:

```
<p class="paragraph">This is a paragraph with a class attribute.</p>
<p id="first-paragraph">This is the first paragraph with an ID attribute.</p>
```

In the first example, the class attribute is set to "paragraph", which can be used to style the paragraph using CSS. In the second example, the ID attribute is set to "first-paragraph", which can be used to create links that point directly to that paragraph. Using paragraphs in HTML can help to organize content on a web page and make it easier to read and understand.

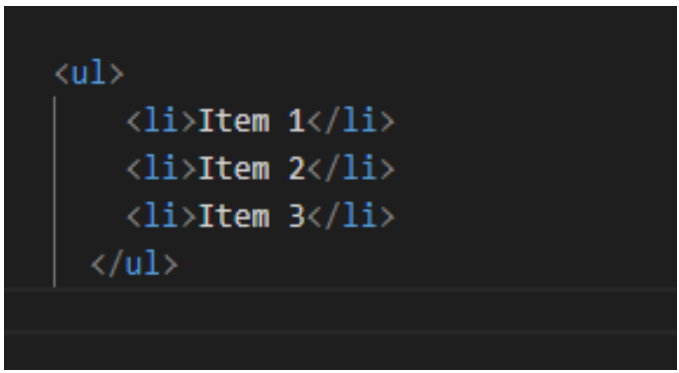
# Understanding HTML Lists

Lists are commonly used in HTML to organize and present information in a structured way. There are two

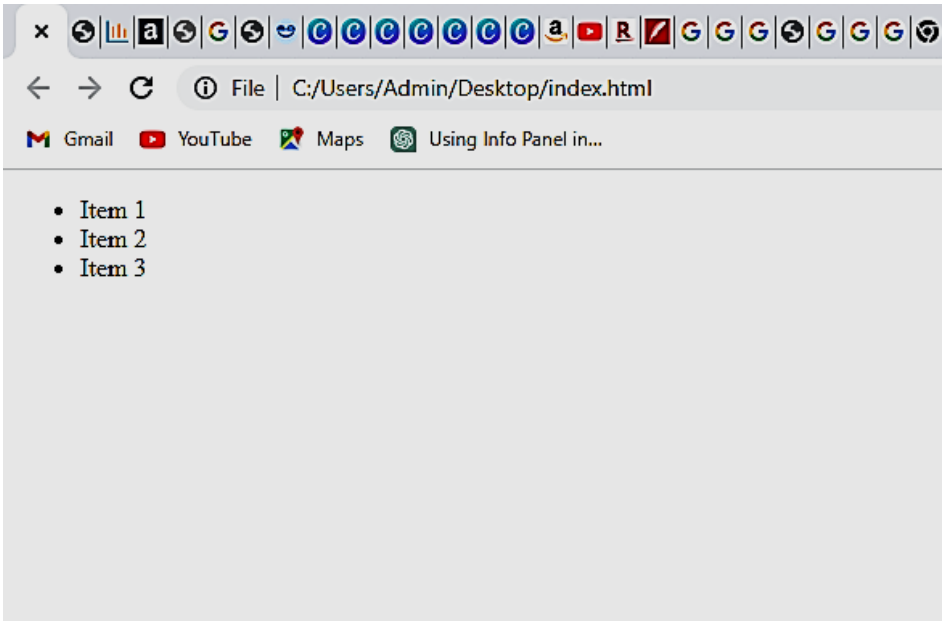
main types of lists in HTML: ordered lists and unordered lists.

#### Unordered Lists

Unordered lists are used to present a list of items in no particular order. To create an unordered list, you can use the <ul> element. Here's an example:




In the example, the <ul> element is used to define the unordered list, and each item in the list is defined using the <li> (list item) element. The result will look like this:




# Ordered Lists

Ordered lists are used to present a list of items in a specific order. To create an ordered list, you can use the <ol> element. Here's an example:

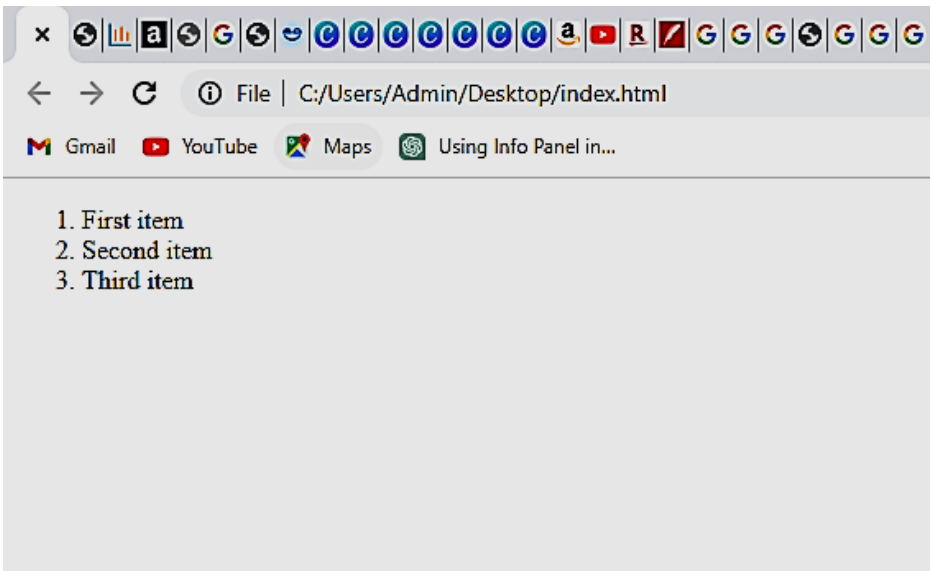
Terminal Help index.html - Visual Studio Code

index.py Document.html index.html

C: > Users > Admin > Desktop > index.html > html > body

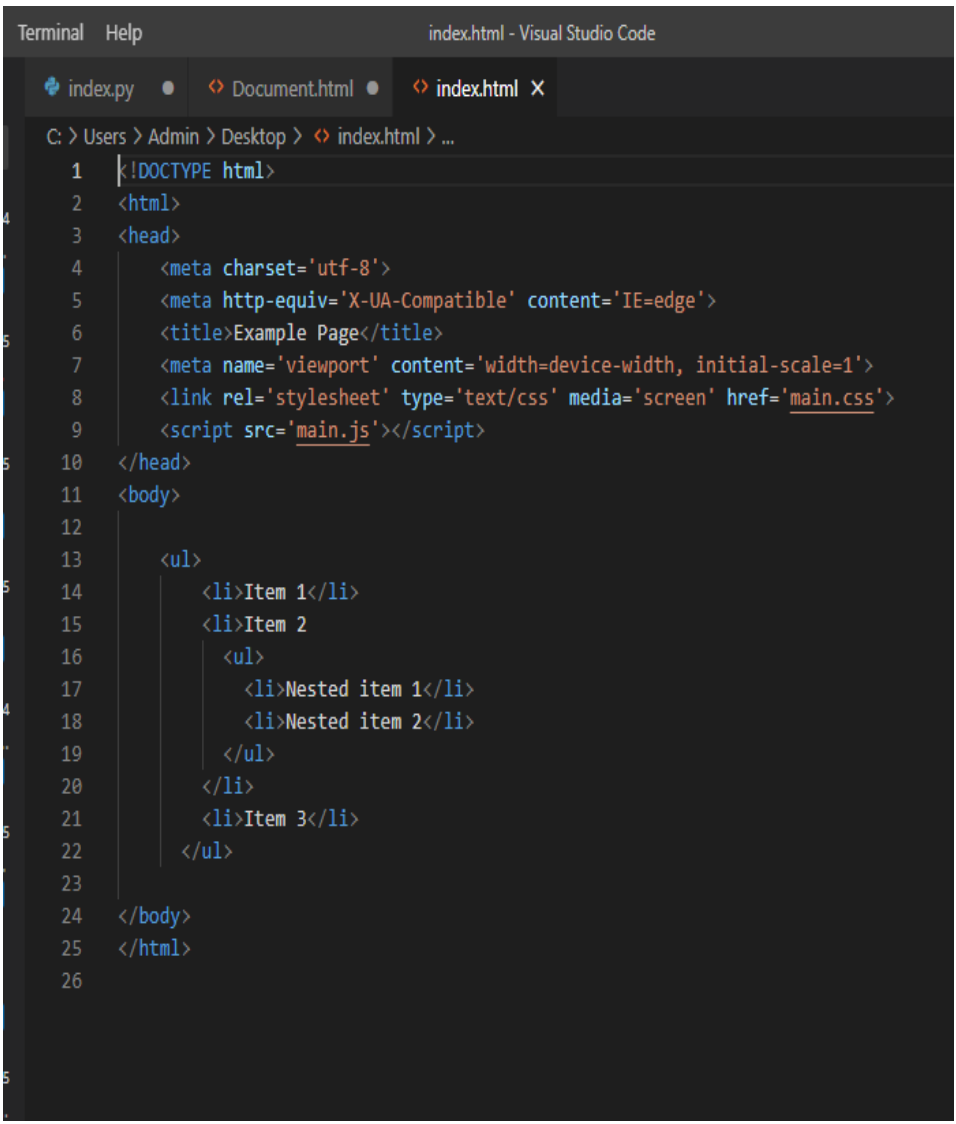
```
<!DOCTYPE html>
<html>
<head>
    <title>Example Page</title>
</head>
<body>

    <ol>
        <li>First item</li>
        <li>Second item</li>
        <li>Third item</li>
    </ol>


    </body>
</html>
```

In this example, the <ol> element is used to define the ordered list, and each item in the list is defined using the <li> element. The result will look like this:



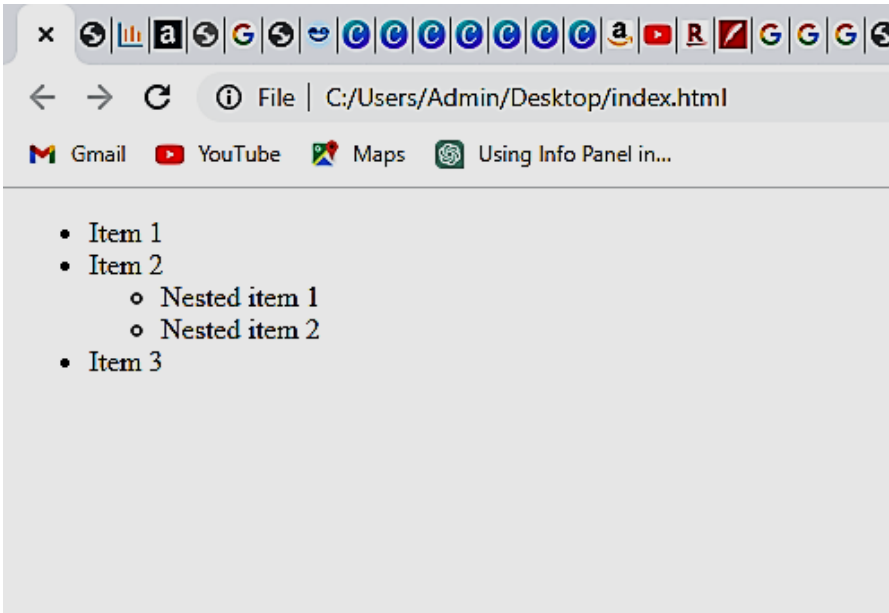
**Nested Lists**

You can also create nested lists in HTML, which means placing one list inside another list. To do this, you can simply place one list inside another using the appropriate tags. Here's an example:




In this example, a nested unordered list is created by placing a new <ul> element inside the second list item.

The result will look like:




Using lists in HTML can help to organize and present information in a clear and structured way, making it easier for users to understand and navigate.

# Understanding HTML links

Links are an important aspect of web pages as they allow users to navigate between pages or to other resources on the web. To create a link in HTML, you can use the <a> element, which stands for anchor. Here's an example of how to create a basic link:

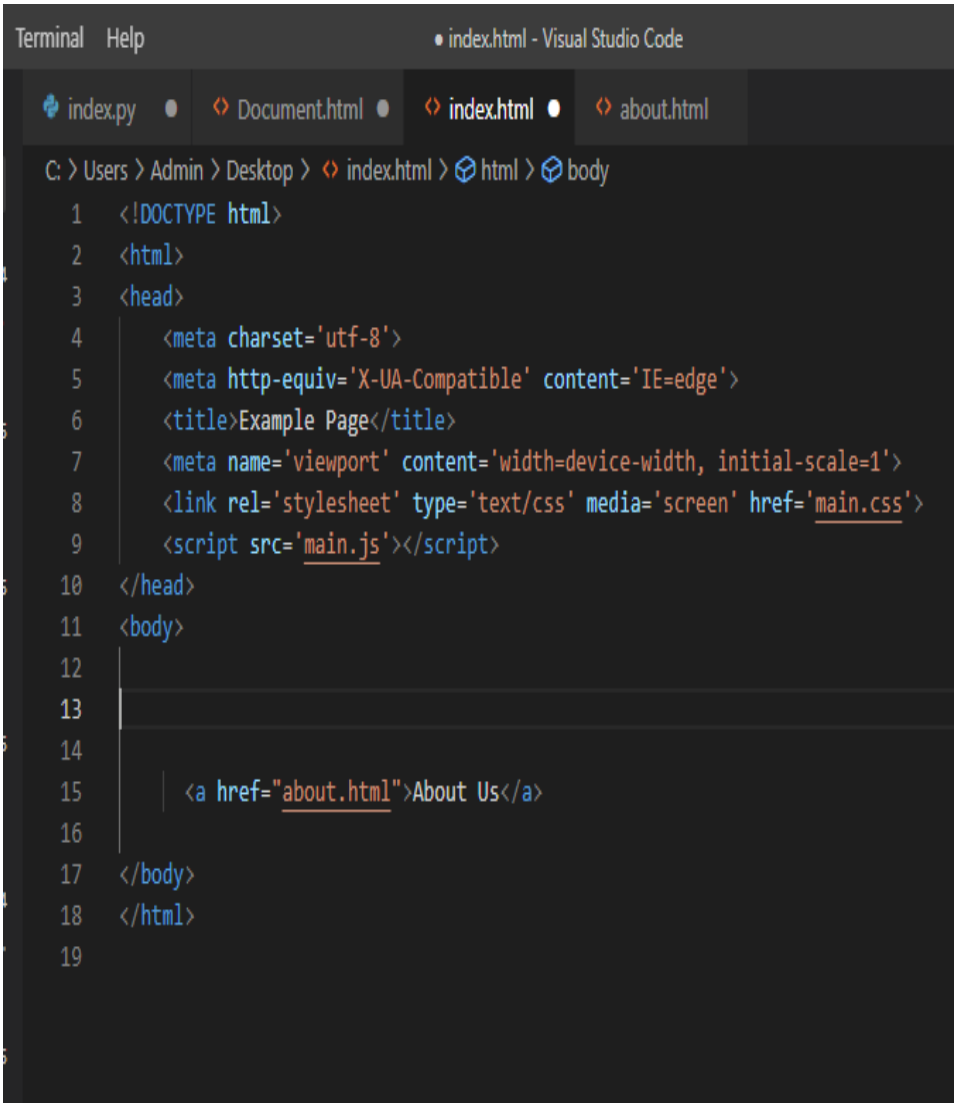
*<a href="http://www.google.com">Link text</a>*

In this example, the href attribute specifies the URL (Uniform Resource Locator) of the destination page or resource, and the link text is the text that the user

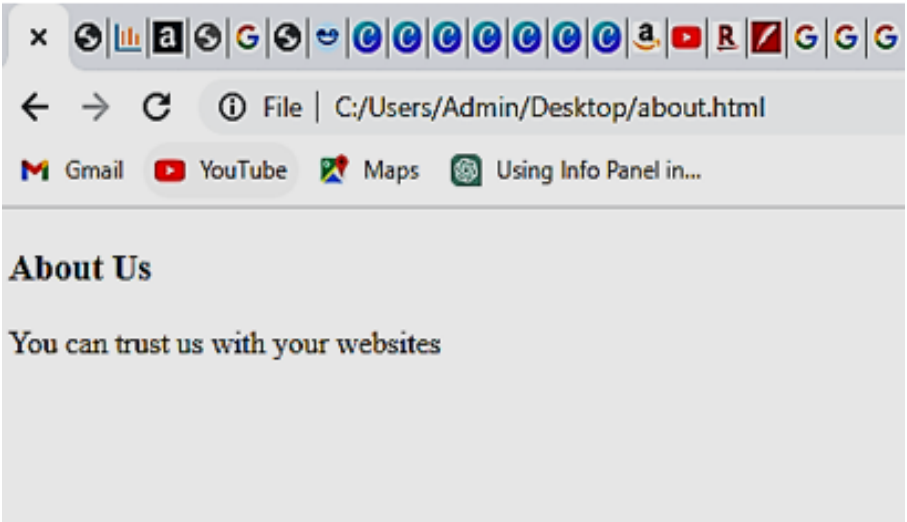
clicks on to activate the link. When the user clicks on the link text, the browser will navigate to the URL specified in the href attribute.

## Relative URLs

In addition to absolute URLs, you can also use relative URLs to link to pages or resources within the same website. Relative URLs specify the path to the page or resource relative to the current page. Here's an example:

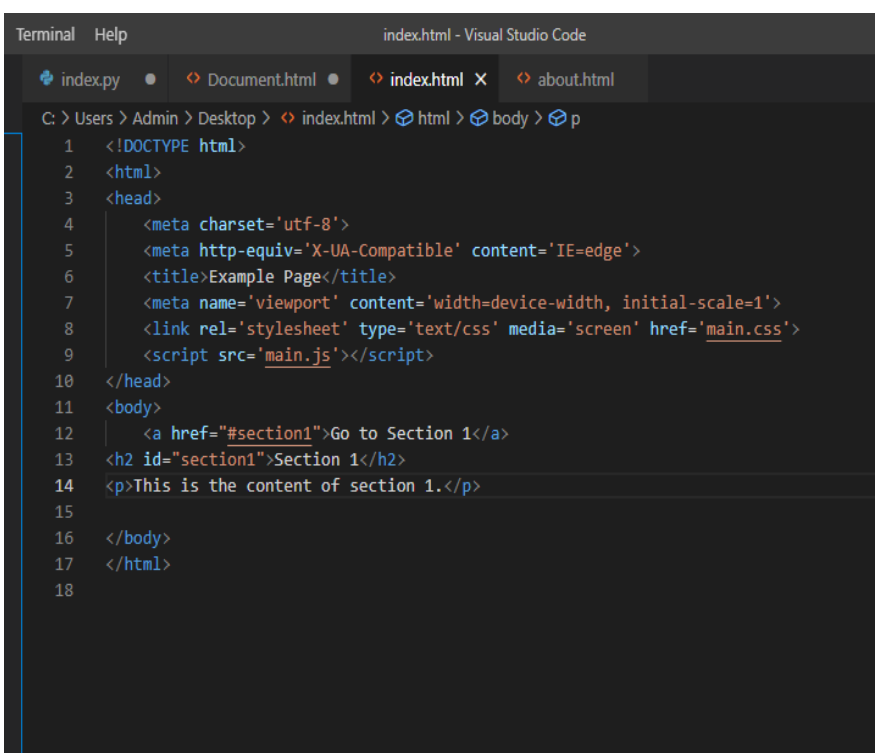



In this example, the link will navigate to the "about.html" page in the same directory as the current page.

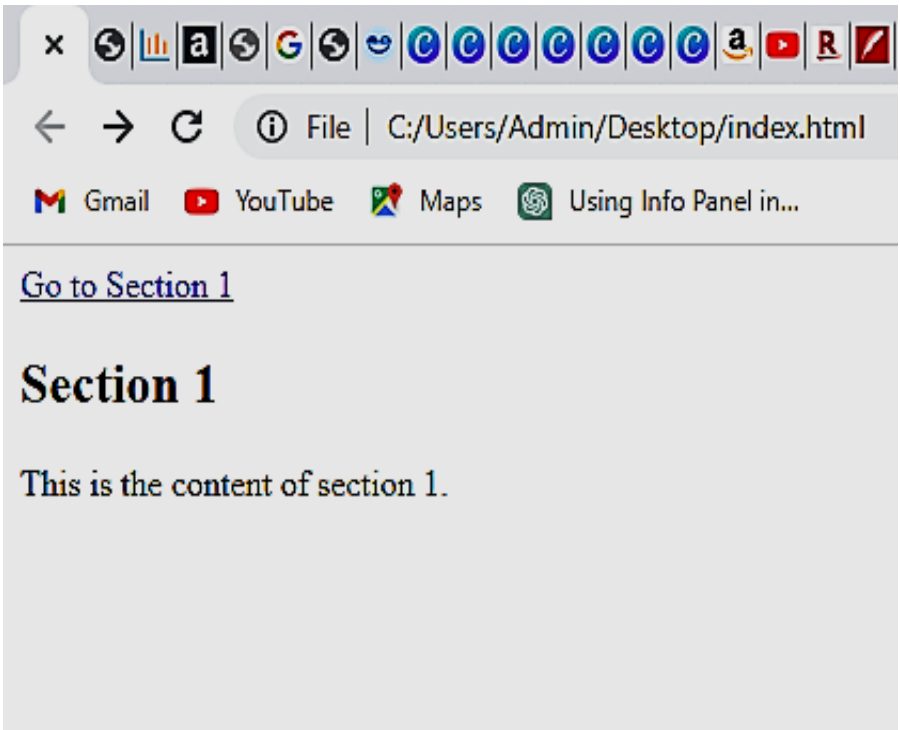



**Linking to Specific Parts of a Page**

You can also create links that navigate to specific parts of a page by using the id attribute. Here's an example:



In this example, the link text "Go to Section 1" will navigate to the section of the page with the id attribute set to "section1". When the link is clicked, the page will scroll to the section and highlight it. Using links in HTML is an important part of creating a user-friendly website. By using descriptive link text and providing clear and relevant destinations, you can help users navigate your website more easily. See the result below:




# How to Format Text using HTML Tags

HTML tags are used to format and structure text on a web page. Here are some common HTML tags and how they are used to format text:

- **Headings**: Headings are used to organize the content of the page and provide a visual hierarchy. There are six levels of headings, from h1 (the most important) to h6 (the least important). To use a heading tag, simply wrap your text in the appropriate tag.
- **Paragraphs**: Paragraphs are used to separate blocks of text. To create a paragraph, simply wrap your text in the <p> tag.
- **Bold and Italic**: You can emphasize text using the  tag for bold text and the  tag for italic text.
- **Lists**: There are two types of lists in HTML: ordered lists (<ol>) and unordered lists (<ul>). To create a list, wrap your list items in the appropriate tags:
- **Links**: To create a hyperlink, use the <a> tag and include the URL in the href attribute.

These are just a few of the most commonly used HTML tags for formatting text. There are many more tags available for formatting text, images, tables, and other elements on a web page.

# How to Add Images to an HTML Document

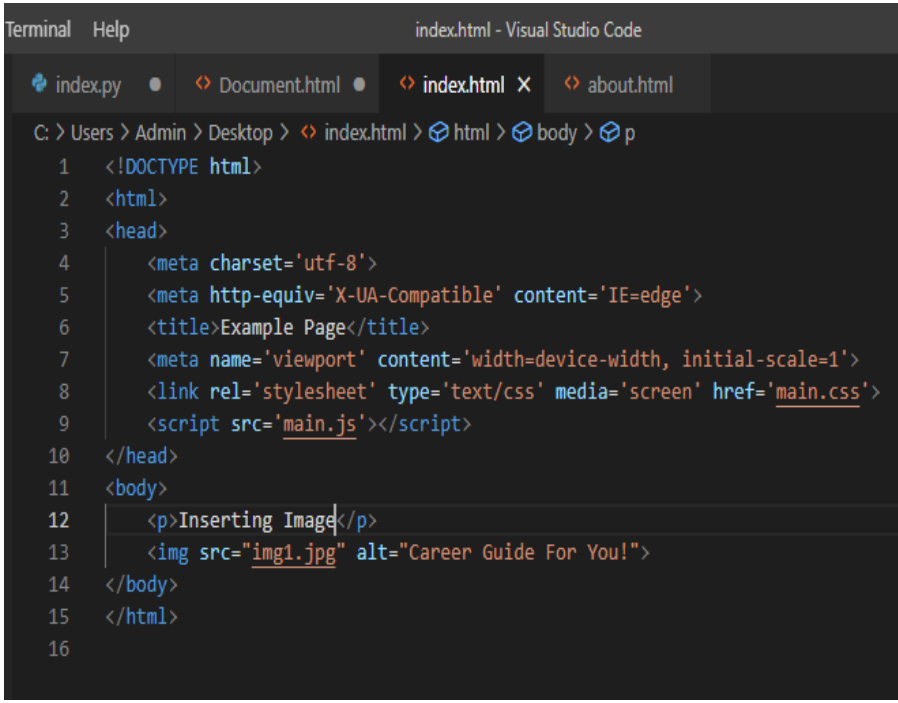
To add an image to an HTML document, you can use the <img> tag. Here's how:

- Save the image file to your computer in a location that you can easily find, such as your desktop.
- Open your HTML file in a text editor or an HTML editor.

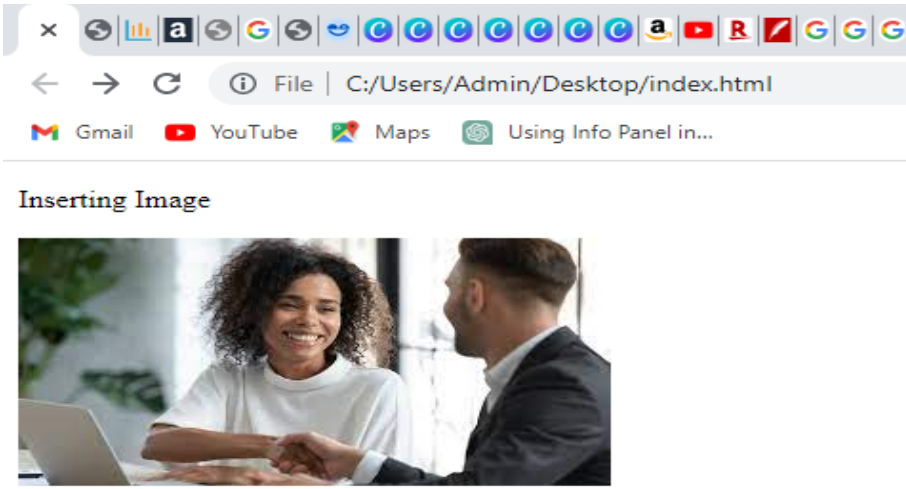
In the HTML file, add a <img> tag where you want the image to appear. The <img> tag has two required attributes:

- src: This specifies the URL of the image file.
- alt: This provides alternative text for the image, which is displayed if the image can't be loaded.

Here's an example of a <img> tag:




In this example, replace the image path with the actual file path of your image on your computer. Also, add a short description of the image. Save the HTML file and open it in your web browser to see the image as follows.




Note that the src attribute can use either a relative or absolute file path to specify the location of the image file. If the image file is in the same directory as the HTML file, you can use a relative file path like this:

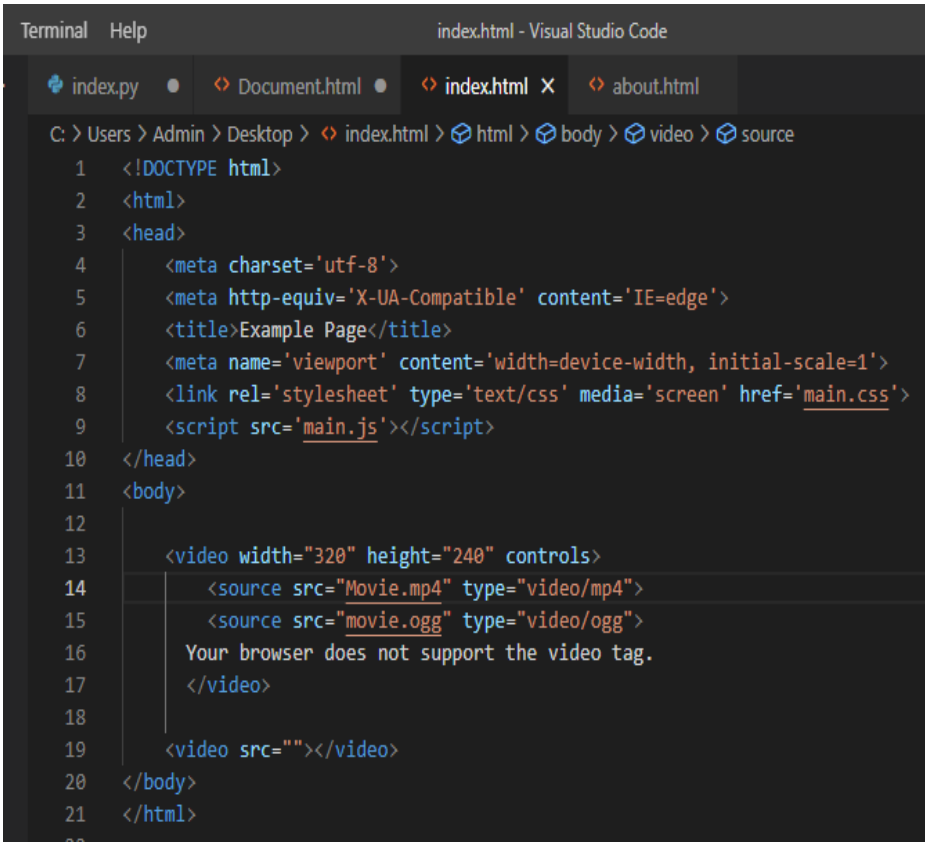
```
<img src="image.jpg" alt="Career Guide for You!">
```

If the image file is in a different directory, you'll need to specify the path to that directory, like this:

In this example, the image file is located in a subdirectory called images within the directory containing the HTML file.

# How to Add Videos to an HTML Document

To add videos to an HTML5 document, you can use the **<video>** element. Here's an example of how to use the **<video>** element:

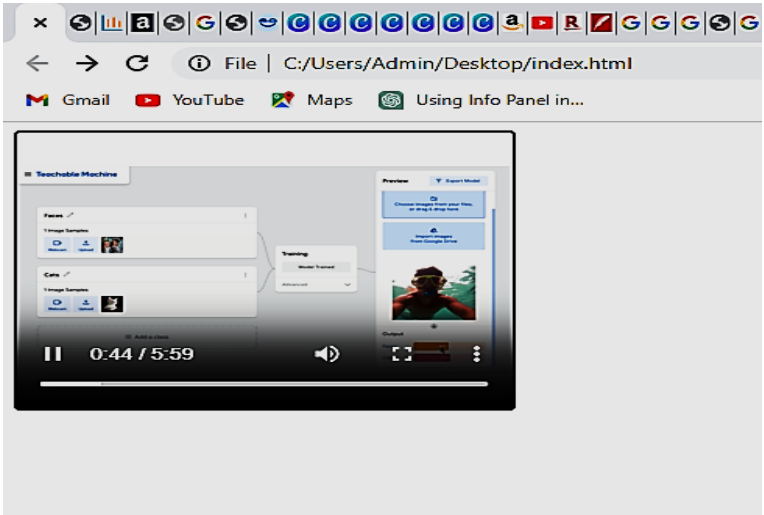



In this example, the **<video>** element creates a video player on the web page, with the **controls** attribute indicating that the player should include basic playback controls such as play, pause, and volume.

The **<source>** element within the **<video>** element specifies the location of the video file, along with its MIME type. In this example, there are two **<source>** elements, one for the MP4 format and one for the WebM format. The browser will use the first format it can support. You can include additional **<source>** elements to support other video formats.

The text within the **<video>** element (in this case, "Your browser does not support the video tag.") is

displayed if the user's browser does not support the **<video>** element or any of the specified video formats. See the output of the above screenshot below:




Here are some additional attributes you can use with the **<video>** element:

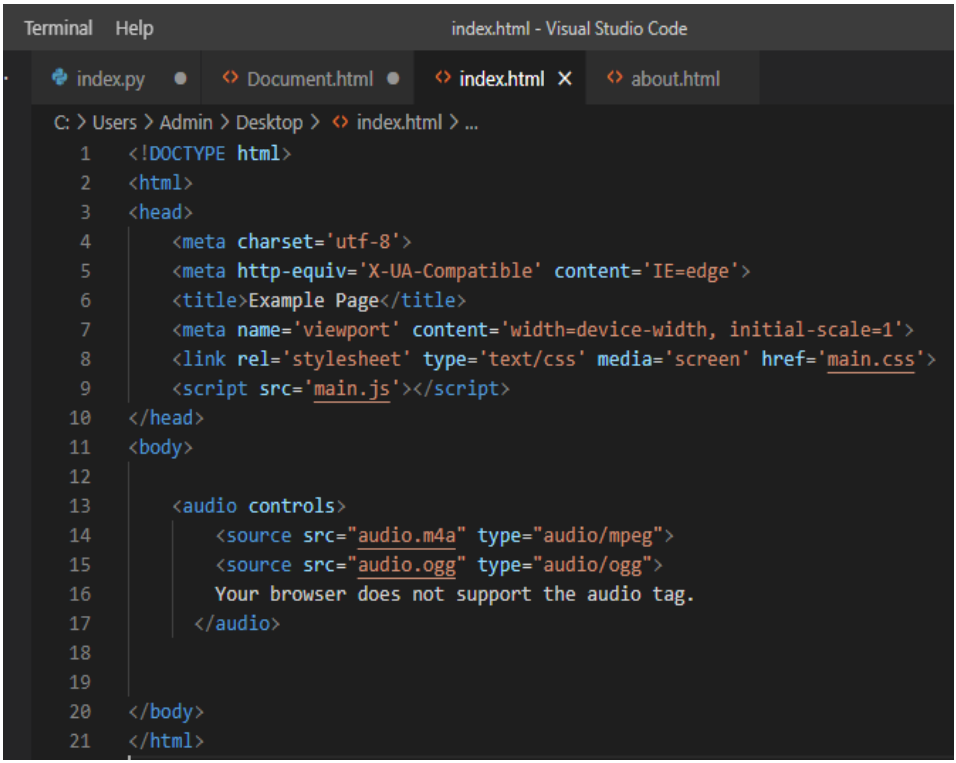
- **width**: Specifies the width of the video player, in pixels.
- **height**: Specifies the height of the video player, in pixels.
- **autoplay**: Specifies that the video should start playing automatically when the web page loads.
- **loop**: Specifies that the video should play in a continuous loop.
- **muted**: Specifies that the video should play without sound.

Keep in mind that video files can be large, so it's important to optimize them for web playback to ensure that they load quickly and don't use too much bandwidth. This can include compressing the video file and using the appropriate video format for your needs.

# Using Audio in HTML5

To embed audio content in an HTML or XHTML document, HTML5 supports the audio> tag. The audio tag in the current HTML5 draft specification does not list the audio formats that browsers should support. Ogg, mp3, and wav are the audio formats that are most frequently used.

Before HTML5, audio files could be introduced to a page by using the bgsound> tag to integrate background sound. The user was unable to silence the sound, which was played while the user was viewing the page. With HTML5, there is no need to link third-party plugins because audio files may be included using the audio> tag. HTML or Javascript can be used to control the audio element, and CSS can be used to style it. To use audio in HTML5, you can use the **<audio>** element. Here's an example of how to use the **<audio>** element:

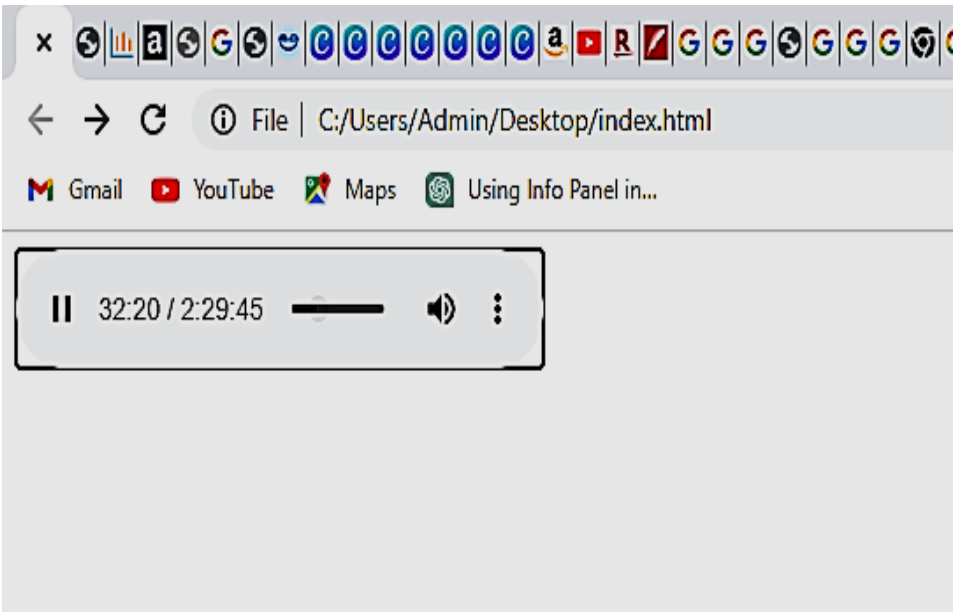



In this example, the **<audio>** element creates an audio player on the web page, with the **controls** attribute indicating that the player should include basic playback controls such as play, pause, and volume.

The **<source>** element within the **<audio>** element specifies the location of the audio file, along with its MIME type. In this example, there are two **<source>** elements, one for the MP3 format and one for the Ogg Vorbis format. The browser will use the first format it can support. You can include additional **<source>** elements to support other audio formats.

The text within the **<audio>** element (in this case, "Your browser does not support the audio tag.") is displayed if the user's browser does not support the

**<audio>** element or any of the specified audio formats. See the result of the code above below.




Here are some additional attributes you can use with the **<audio>** element:

- **autoplay**: Specifies that the audio should start playing automatically when the web page loads.
- **loop**: Specifies that the audio should play in a continuous loop.
- **muted**: Specifies that the audio should play without sound.

You can also use JavaScript to control the playback of the audio, for example, to add additional controls or to change the playback speed.

# CHAPTER THREE

# UNDERSTANDING FORMS AND INPUT

Understanding forms and input is an essential part of web development. Forms allow users to submit information to a website, whether it's creating an account, signing up for a newsletter, or making a purchase. In this chapter, we will explore the basics of creating forms and the different types of form elements, such as text input fields, radio buttons, checkboxes, and select menus. We will also discuss how to style form elements to create a cohesive and visually appealing user interface. Understanding forms and input is crucial for creating effective and user-friendly web applications, so let's dive in!

## Creating Forms

Creating forms involves using several elements and attributes to allow users to input data or information that can be submitted to a server for processing. Here are the basic steps to create a form in HTML5:

1. Use the **<form>** element to create a form on the web page. The **action** attribute specifies the URL where the form data will be submitted, and the **method** attribute specifies the HTTP method to use when submitting the form data (either **GET** or **POST**).

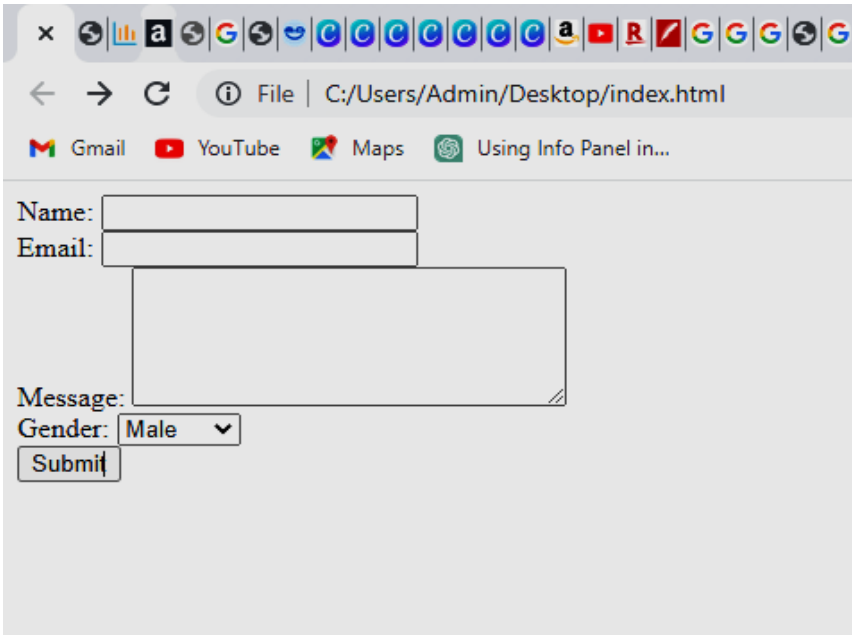
```
<body>

    <form action="/submit-form" method="POST">
        <!-- Form elements go here -->
    </form>

</body>
</html>
```

1. Use form elements such as **<input>**, **<label>**, **<textarea>**, **<select>**, and **<button>** to create form fields, labels, and buttons.

Example:

```
<body>

    <form action="/submit-form" method="POST">
        <label for="name">Name:</label>
        <input type="text" id="name" name="name" required>
        <br>

        <label for="email">Email:</label>
        <input type="email" id="email" name="email" required>
        <br>

        <label for="message">Message:</label>
        <textarea id="message" name="message" rows="5" cols="30"></textarea>
        <br>

        <label for="gender">Gender:</label>
        <select id="gender" name="gender">
            <option value="male">Male</option>
            <option value="female">Female</option>
        </select>
        <br>

        <button type="submit">Submit</button>
    </form>

</body>
</html>
```

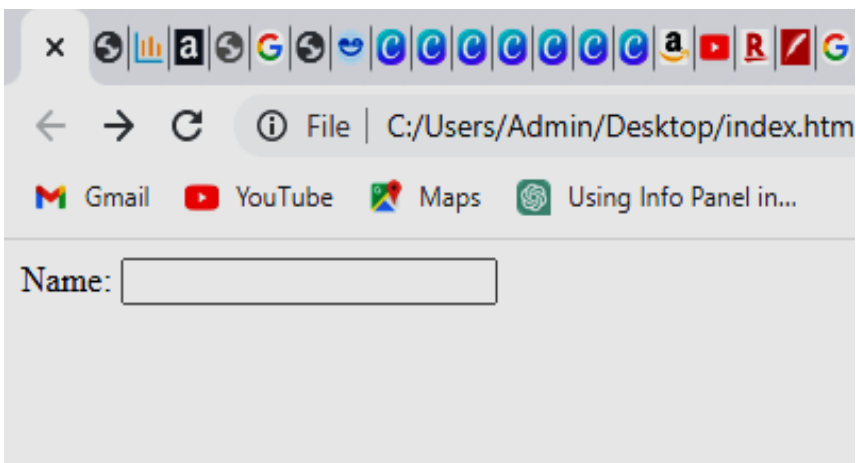
In this example, the form includes several form elements such as text input fields for name and email, a text area for the message, a dropdown list for gender, and a submit button. Each form element has its unique attributes, such as **type**, **id**, **name**, and **value**. See the output below:




1. Use the **for** attribute in the **<label>** element to link the label to its corresponding form element using the **id** attribute. This improves accessibility and user experience by making it easier to select the correct form element.

```
    <form action="/submit-form" method="POST">
        <label for="name">Name:</label>
        <input type="text" id="name" name="name" required>
        <br>
        <!-- More form elements -->
    </form>


</body>
</html>
```

See the output below.




1. Use the **required** attribute on the form element to make it mandatory for the user to enter a value in the form field before submitting the form.

Example:

```
<body>

    <form action="/submit-form" method="POST">
        <label for="name">Name:</label>
        <input type="text" id="name" name="name" required>
        <br>
        <!-- More form elements -->
    </form>

</body>
</html>
```

1. Use the **type** attribute in the **<input>** element to specify the type of input field you want to use, such as text, email, password, radio buttons, checkboxes, etc.

Example:

```
    <form action="/submit-form" method="POST">
        <label for="email">Email:</label>
        <input type="email" id="email" name="email" required>
        <br>
        <!-- More form elements -->
    </form>


</body>
</html>
```

## Text Inputs and Labels

In HTML, text inputs and labels are used to create user input fields on a web page. Text inputs allow users to enter text, while labels are used to describe

what type of information should be entered into the input field.

To create a text input field, you can use the **<input>** tag with the **type="text"** attribute. For example:

```
<body>

    <label for="name">Name:</label>
<input type="text" id="name" name="name">


</body>
</html>
```

In the above code, the **label** tag is used to describe what information should be entered into the input field. The **for** attribute in the label tag is used to associate the label with the input field using the **id** attribute. The **id** attribute is used to uniquely identify the input field. The **name** attribute is used to give the input field a name that can be used when the form is submitted.

To create a label, you can use the **<label>** tag. The **for** attribute in the label tag should be set to the **id** attribute of the corresponding input field. This helps screen readers and other accessibility tools associate the label with the input field.

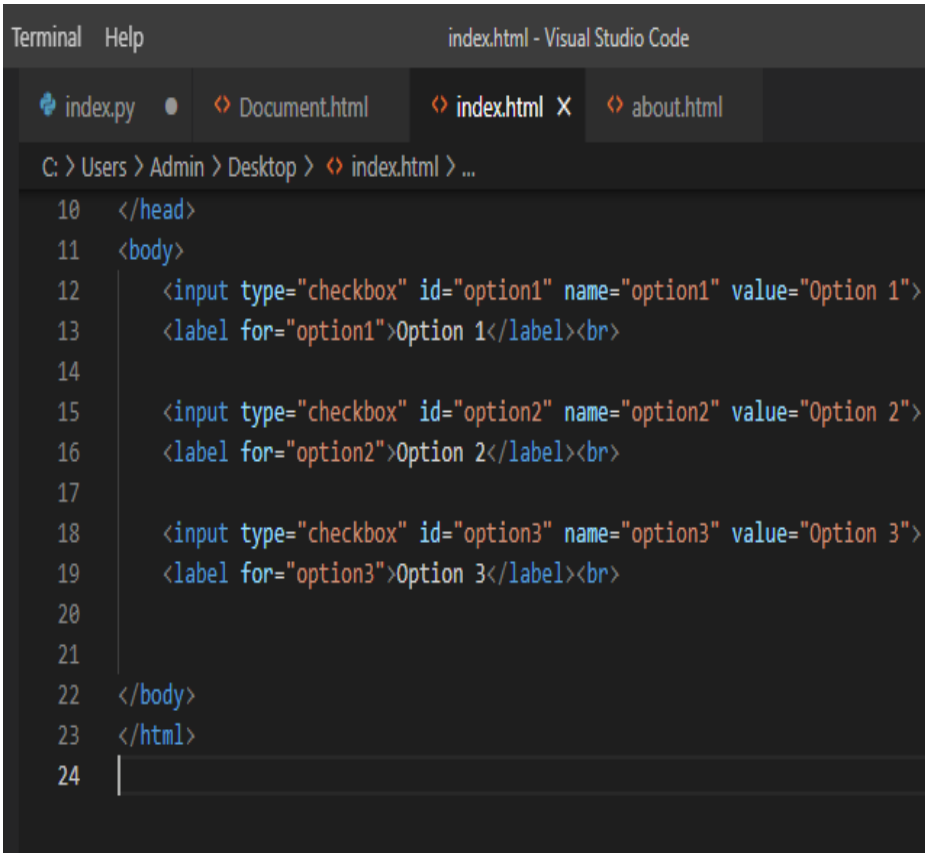
```
<body>

    <label for="email">Email:</label>
    <input type="email" id="email" name="email">

</body>
</html>
```

In the above code, the **type** attribute is set to "email" to ensure that the user enters a valid email address. You can also use other input types, such as "password", "number", "date", and more, to create input fields that accept different types of data.

### Checkboxes and Radio Buttons

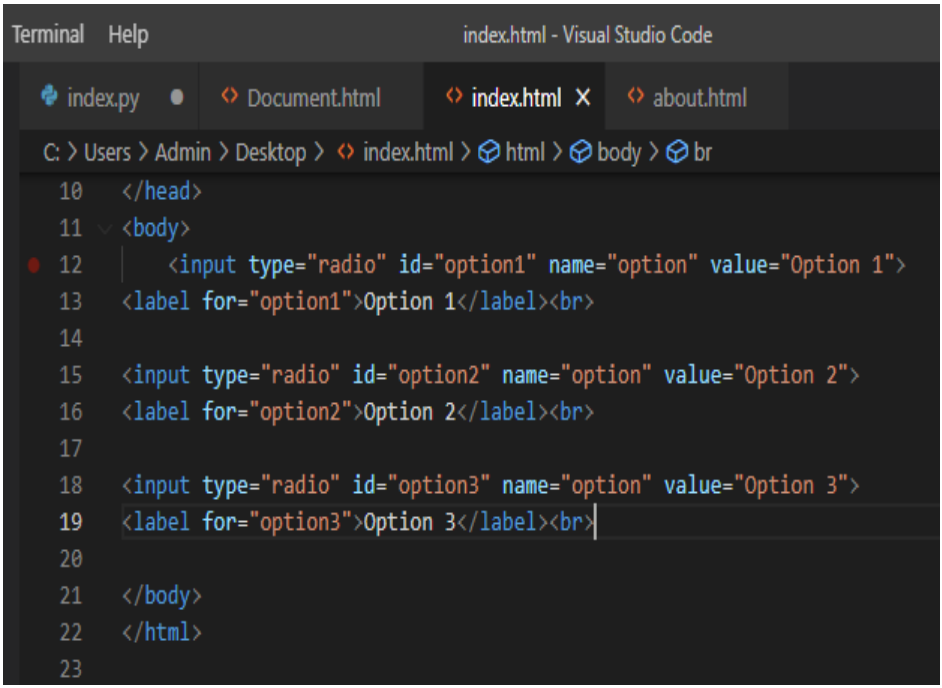
Checkboxes and radio buttons are two types of input fields that are commonly used in web forms to allow users to make selections from a list of options.

A checkbox allows the user to select one or more options from a list of choices. To create a checkbox in HTML, you can use the **<input>** tag with the **type="checkbox"** attribute.




In the above code, each checkbox has a unique **id** and **name** attribute, which allows the server to process the user's selections. The **value** attribute is used to define the value associated with each checkbox when the form is submitted.

A radio button, on the other hand, allows the user to select only one option from a list of choices. To create a radio button in HTML, you can use the **<input>** tag with the **type="radio"** attribute.




In the above code, each radio button has the same **name** attribute, which ensures that only one option can be selected at a time. When the form is submitted, the value of the selected radio button is sent to the server.

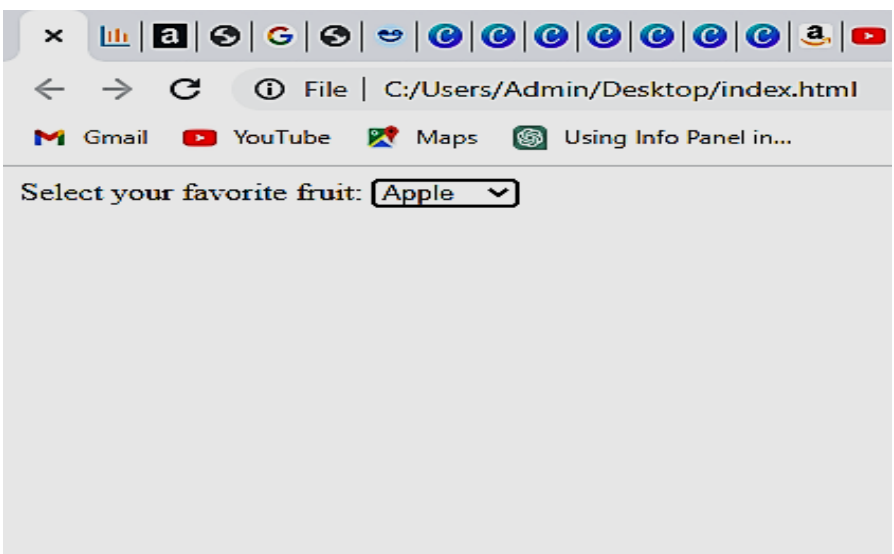
## Select Menus and Option Groups

Select menus and option groups are two more types of input fields that are commonly used in web forms to allow users to make selections from a list of options.

A select menu is a drop-down list of options that the user can choose from. To create a select menu in HTML, you can use the **<select>** tag with one or more **<option>** tags inside.

```
<body>
    <label for="fruit">Select your favorite fruit:</label>
    <select id="fruit" name="fruit">
      <option value="apple">Apple</option>
      <option value="banana">Banana</option>
      <option value="orange">Orange</option>
      <option value="mango">Mango</option>
    </select>

</body>
</html>
```

See the output below.



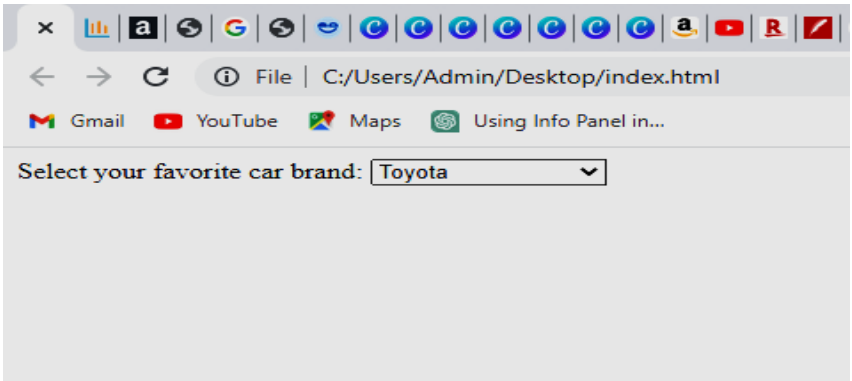

In the above code, the **id** and **name** attributes are used to uniquely identify the select menu. Each **<option>** tag has a **value** attribute that defines the

value associated with that option when the form is submitted.

Option groups are used to group related options together in a select menu. To create an option group in HTML, you can use the **<optgroup>** tag with one or more **<option>** tags inside.

```
<body>
    <label for="cars">Select your favorite car brand:</label>
<select id="cars" name="cars">
  <optgroup label="Japanese Cars">
    <option value="toyota">Toyota</option>
    <option value="honda">Honda</option>
    <option value="nissan">Nissan</option>
  </optgroup>
  <optgroup label="European Cars">
    <option value="bmw">BMW</option>
    <option value="mercedes">Mercedes-Benz</option>
    <option value="audi">Audi</option>
  </optgroup>
</select>

</body>
</html>
```

See the output below.




In the above code, the **<optgroup>** tag is used to group the options into two categories: Japanese Cars and European Cars. The **label** attribute is used to define the name of each option group.

## Text Areas and Buttons

Text areas and buttons are two additional types of input fields that are commonly used in web forms to allow users to provide more detailed information or to submit the form data.

A text area is a multi-line input field that allows the user to enter a larger amount of text. To create a text area in HTML, you can use the **<textarea>** tag with the **cols** and **rows** attributes to define the size of the input field.

```
<body>
    <label for="message">Enter your message:</label>
<textarea id="message" name="message" rows="5" cols="30"></textarea>


</body>
</html>
```

In the above code, the **id** and **name** attributes are used to uniquely identify the text area, and the **rows** and **cols** attributes are used to define the number of rows and columns in the text area.

A button is an input field that the user can click to submit the form or perform some other action. There are three types of buttons in HTML: submit, reset, and button. The submit button is used to submit the form data, the reset button is used to reset the form to its initial state, and the button type is used for custom actions.

```
<body>
    <button type="submit">Submit</button>
<button type="reset">Reset</button>
<button type="button">Click me</button>

</body>
</html>
```

In the above code, each button has a **type** attribute that specifies the type of button. The text inside the button tags is the label that appears on the button.

### Understanding Form Validation

Form validation is the process of checking the user input in a form to ensure that it meets certain requirements or constraints. Validation is important because it helps to ensure the accuracy and consistency of the data collected through web forms.

HTML5 provides built-in form validation features that allow you to specify validation requirements for input fields using attributes. The validation is done automatically in modern web browsers, and the user will be prompted with error messages if the input does not meet the specified requirements.

Some of the common attributes that are used for form validation in HTML5 include:

- **required**: Specifies that the input field must be filled out before the form can be submitted.
- **pattern**: Specifies a regular expression pattern that the input must match.
- **min** and **max**: Specifies the minimum and maximum allowed values for a numeric input field.
- **minlength** and **maxlength**: Specifies the minimum and maximum lengths of a text input field.
- **type**: Specifies the type of input, such as email, url, number, or date, and the browser will validate the input accordingly.

Here is an example of how to use some of these validation attributes:

```
<body>
    <form>
        <label for="username">Username:</label>
        <input type="text" id="username" name="username" required minlength="5">

        <label for="email">Email:</label>
        <input type="email" id="email" name="email" required>

        <label for="password">Password:</label>
        <input type="password" id="password" name="password" required minlength="8">

        <label for="age">Age:</label>
        <input type="number" id="age" name="age" min="18" max="99">

        <button type="submit">Submit</button>
      </form>

</body>
</html>
```

In the above code, the **required** attribute is used to ensure that the username, email, and password fields are filled out before the form can be submitted. The **minlength** attribute is used to specify that the username and password fields must be at least 5 and 8 characters long, respectively. The **type** attribute is used to specify the email and number input fields and to enable the browser to validate them accordingly.

## CHAPTER FOUR

## UNDERSTANDING TABLES AND LISTS

Tables and lists are important HTML elements used to organize and present information in a structured and readable format on web pages. Tables are commonly used to display data in rows and columns, while lists are used to display content in a bulleted or numbered format.

In this chapter, we will explore how to create and style tables and lists in HTML. We will also cover some of the advanced features of tables, such as colspan and rowspan, and show how to create nested lists. Additionally, we will discuss some best practices for designing and formatting tables and lists to ensure they are accessible and easy to read for all users.

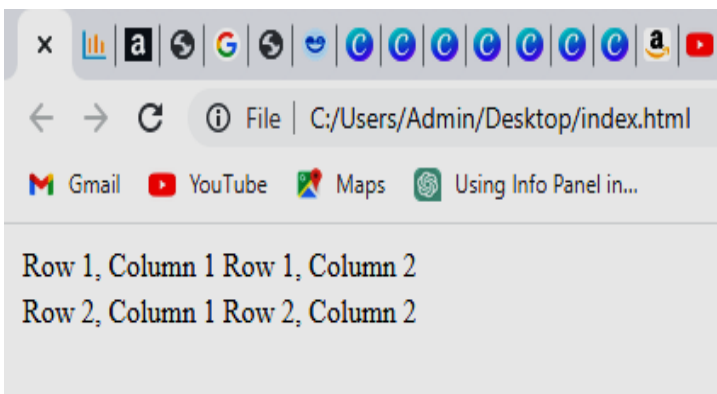
By the end of this chapter, you will have a good understanding of how to use tables and lists to organize and present content on your web pages. You will also be able to apply various styling techniques to enhance the visual appearance of your tables and lists.

# Creating Tables

To create a table in HTML, you will need to use the **<table>** element. Within the table element, you can use other elements to create rows and cells. Here is a basic example of how to create a simple table with two rows and two columns:

```
<body>
    <table>
        <tr>
            <td>Row 1, Column 1</td>
            <td>Row 1, Column 2</td>
        </tr>
        <tr>
            <td>Row 2, Column 1</td>
            <td>Row 2, Column 2</td>
        </tr>
      </table>

</body>
</html>
```

See the result below.



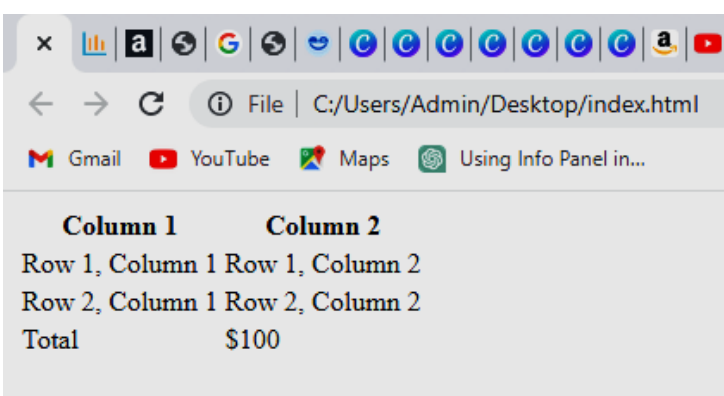

In the above code, we have created a table with two rows and two columns. The **<tr>** element represents a table row, and the **<td>** element represents a table cell. The text within each **<td>** element will be displayed in the corresponding cell in the table.

By default, the table cells will be separated by borders, but you can use CSS to change the border style, color, and thickness. You can also add other elements such

as **<thead>**, **<tbody>**, and **<tfoot>** to further structure your table and add headers or footers.

```
<body>
    <table>
        <thead>
          <tr>
            <th>Column 1</th>
            <th>Column 2</th>
          </tr>
        </thead>
        <tbody>
          <tr>
            <td>Row 1, Column 1</td>
            <td>Row 1, Column 2</td>
          </tr>
          <tr>
            <td>Row 2, Column 1</td>
            <td>Row 2, Column 2</td>
          </tr>
        </tbody>
        <tfoot>
          <tr>
            <td>Total</td>
            <td>$100</td>
          </tr>
        </tfoot>
      </table>

</body>
</html>
```

See the result of the code below.



In this example, we have added an **<thead>** element to create a table header with two columns, and a **<tfoot>** element to create a table footer with a summary of the data. The **<tbody>** element contains the actual data rows of the table.

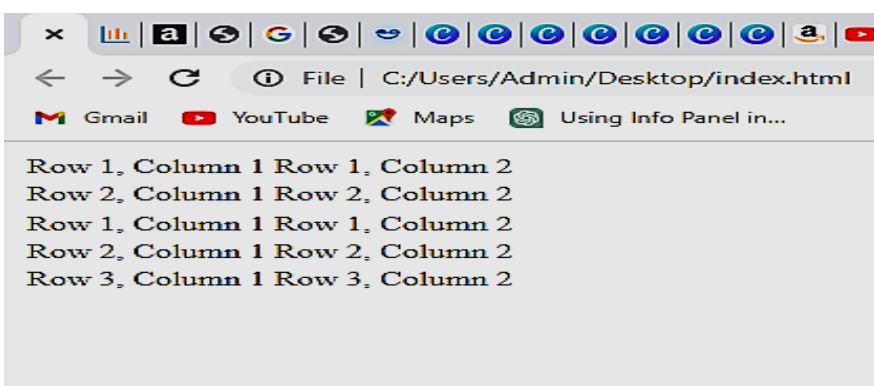
Tables can also be made more complex with the use of attributes such as **colspan** and **rowspan**, which allow cells to span multiple columns or rows. Additionally, you can use CSS to style the table with borders, background colors, and font styles.

### Adding Rows and Columns

To add rows and columns to an existing table in HTML, you can use the **<tr>** and **<td>** elements within the table. Here's an example of how to add a row to an existing table:

```
<body>
    <table>
        <tr>
            <td>Row 1, Column 1</td>
            <td>Row 1, Column 2</td>
        </tr>
        <tr>
            <td>Row 2, Column 1</td>
            <td>Row 2, Column 2</td>
        </tr>
    </table>

    <!-- Adding a row -->
    <table>
        <tr>
            <td>Row 1, Column 1</td>
            <td>Row 1, Column 2</td>
        </tr>
        <tr>
            <td>Row 2, Column 1</td>
            <td>Row 2, Column 2</td>
        </tr>
        <tr>
            <td>Row 3, Column 1</td>
            <td>Row 3, Column 2</td>
        </tr>
    </table>
</body>
</html>
```

See the result of the code above below.


File | C:/Users/Admin/Desktop/index.html
Gmail
YouTube
Maps
Using Info Panel in...
Row 1, Column 1 Row 1, Column 2
Row 2, Column 1 Row 2, Column 2
Row 1, Column 1 Row 1, Column 2
Row 2, Column 1 Row 2, Column 2
Row 3, Column 1 Row 3, Column 2

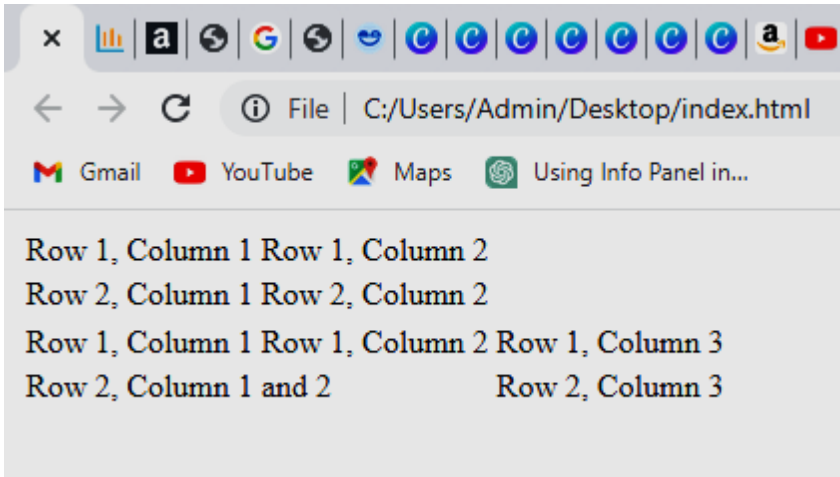
In this example, we have added a new row to the existing table by inserting a new **<tr>** element with two

**<td>** elements containing the text for the new row.

To add a column to an existing table, you can use the **colspan** attribute to make a cell span multiple columns:

```
<body>
    <table>
        <tr>
          <td>Row 1, Column 1</td>
          <td>Row 1, Column 2</td>
        </tr>
        <tr>
          <td>Row 2, Column 1</td>
          <td>Row 2, Column 2</td>
        </tr>
      </table>

      <!-- Adding a column -->
      <table>
        <tr>
          <td>Row 1, Column 1</td>
          <td>Row 1, Column 2</td>
          <td>Row 1, Column 3</td>
        </tr>
        <tr>
          <td colspan="2">Row 2, Column 1 and 2</td>
          <td>Row 2, Column 3</td>
        </tr>
      </table>

</body>
</html>
```

See the result of the code above below.


File | C:/Users/Admin/Desktop/index.html
Gmail
YouTube
Maps
Using Info Panel in...
Row 1, Column 1 Row 1, Column 2
Row 2, Column 1 Row 2, Column 2
Row 1, Column 1 Row 1, Column 2 Row 1, Column 3
Row 2, Column 1 and 2
Row 2, Column 3

In this example, we have added a new column to the existing table by adding a new **<td>** element to each row and then used the **colspan** attribute to make the second cell in the second-row span two columns. This creates a table with three columns, where the second cell in the second row spans two columns.

# Styling Tables with CSS

You can style tables in HTML using CSS to change their appearance, layout, and behavior. Here are some examples of how to style tables with CSS:

- Changing the background color of the table:

```
table {
  background-color: #f2f2f2;
}
```

- Changing the font size and color of the table headers:

```
    th {
  font-size: 18px;
  color: blue;
}
```

- Setting a fixed width for the table and its columns:

```
    table {
  width: 100%;
  table-layout: fixed;
}

th, td {
  width: 25%;
}
```

- Adding borders and spacing to the table cells:

```
    table {
  border-collapse: collapse;
  border-spacing: 0;
}

td {
  border: 1px solid black;
  padding: 10px;
}
```

- Styling alternating rows with different background colors:

```
    tr:nth-child(even) {
  background-color: #f2f2f2;
}

tr:nth-child(odd) {
  background-color: #fff;
}
```

**Creating Ordered and Unordered Lists**

In HTML, you can create both ordered lists and unordered lists using the **<ol>** and **<ul>** tags, respectively. Here's an example of how to create an unordered list:

```
<body>
    <ul>
        <li>Item 1</li>
        <li>Item 2</li>
        <li>Item 3</li>
    </ul>

</body>
</html>
```

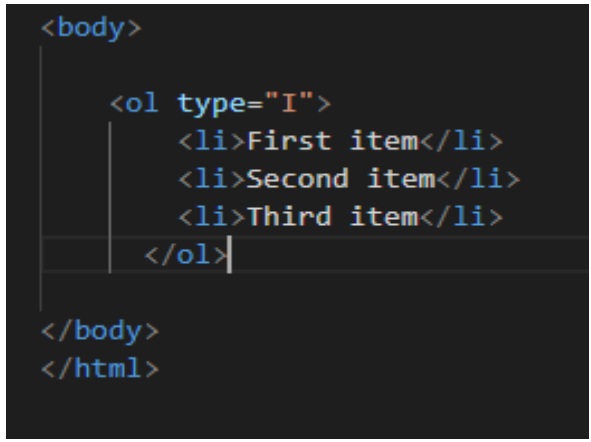
And here's an example of how to create an ordered list:

```
<body>
    <ol>
        <li>First item</li>
        <li>Second item</li>
        <li>Third item</li>
    </ol>

</body>
</html>
```

You can also nest lists inside other lists to create more complex structures. Here's an example of a nested list:

```
<body>

    <ul>
        <li>Item 1</li>
        <li>Item 2
            <ul>
                <li>Subitem 1</li>
                <li>Subitem 2</li>
            </ul>
        </li>
        <li>Item 3</li>
    </ul>

</body>
</html>
```

When creating lists, you can also use the **type** attribute to specify the type of list marker for ordered lists. For example, you can use the **type** attribute to create a Roman numeral list like this:




You can also use CSS to style lists, for example by changing the color or size of the markers, or by adding background colors or borders to the list items.

CHAPTER FIVE

# ADVANCED HTML5 FEATURES

Advanced HTML5 features are powerful tools that can be used to create complex and dynamic web applications. These features include things like canvas, web sockets, web storage, geolocation, and more. In this chapter, we'll take a closer look at some of these advanced features and how they can be used to enhance the functionality and interactivity of your web pages.

Canvas, for example, is a feature that allows you to create dynamic graphics and animations directly within your web page. Web sockets provide a way for real-time communication between a web browser and a server, enabling features like chat applications and online games. Web storage allows you to store data on the client-side, improving performance and reducing the amount of data sent back and forth between the server and client.

By understanding and utilizing these advanced HTML5 features, you can create rich and engaging web applications that provide a better user experience and drive greater engagement and interaction with your content. In this chapter, we'll explore these features in more detail and show you how to use them effectively in your web development projects.

# Canvas and SVG graphics

Canvas and SVG are two popular ways of creating graphics in HTML5. Canvas is a powerful feature that allows you to draw and manipulate graphics directly within your web page using JavaScript. With Canvas, you can create dynamic animations, games, interactive charts, and more. It works by providing a bitmap canvas on which you can draw shapes, images, and other elements using JavaScript. The canvas can then be animated and manipulated in real-time, making it a powerful tool for creating dynamic and interactive web applications.

SVG (Scalable Vector Graphics) is a markup language for describing two-dimensional vector graphics. Unlike Canvas, SVG is not a bitmap-based technology and allows you to create images that are infinitely scalable without loss of quality. With SVG, you can create complex graphics, such as charts, diagrams, logos, and even animations, by using XML-based code. SVG images are also interactive, allowing you to add links, buttons, and other elements to create engaging and interactive user experiences.

While both Canvas and SVG can be used to create graphics in HTML5, they are fundamentally different technologies with different use cases. Canvas is best suited for creating dynamic and interactive graphics that require real-time manipulation, such as games and animations. On the other hand, SVG is ideal for creating high-quality, scalable vector graphics that can be used in a variety of contexts, such as logos, charts,

and diagrams.

## How to Apply Canva and SVG Graphics

To apply Canvas graphics in HTML5, you will need to create a Canvas element in your HTML document, and then use JavaScript to draw shapes, images, and other elements on the canvas. Here's a basic example of how to create a Canvas element:

```
<canvas id="myCanvas" width="500" height="500"></canvas>
```

In the example above, we create a Canvas element with an ID of "myCanvas" and set its width and height to 500 pixels. To draw on the canvas, you can use JavaScript to access the canvas context and use drawing methods to create shapes, lines, and other elements. Here's an example of how to draw a rectangle on the canvas:

```
    var canvas = document.getElementById("myCanvas");
var ctx = canvas.getContext("2d");
ctx.fillStyle = "red";
ctx.fillRect(50, 50, 100, 100);
```

In this example, we use JavaScript to get the Canvas element by its ID and then access its context using the **getContext()** method. We then set the fill style to red

and use the **fillRect()** method to draw a rectangle on the canvas with a position of (50, 50) and a width and height of 100 pixels.

To apply SVG graphics in HTML5, you can create an SVG element in your HTML document and then use SVG markup language to define shapes, lines, and other elements. Here's a basic example of how to create an SVG element:

```
<svg width="500" height="500">
  <rect x="50" y="50" width="100" height="100" fill="red"/>
</svg>
```

In this example, we create an SVG element with a width and height of 500 pixels. We then use the SVG **rect** element to draw a rectangle with a position of (50, 50) and a width and height of 100 pixels, with the fill color set to red.

Both Canvas and SVG graphics can be styled with CSS to further customize their appearance, and both can be animated using JavaScript to create dynamic and interactive user experiences.

# Understanding Web Storage

Web storage is a feature of modern web browsers that allows web developers to store key-value pairs of data

in a user's browser. This data can be stored either temporarily (session storage) or permanently (local storage). Web storage provides a way for web developers to create more responsive and interactive web applications that can store data on the client-side.

Session storage stores data for a specific browsing session and the data is cleared when the user closes the browser window. This type of storage is useful for storing temporary data that is needed for the current session, such as a user's preferences for a particular web application.

Local storage, on the other hand, stores data permanently in the browser until it is cleared by the user or the web application. This type of storage is useful for storing data that needs to persist across multiple browsing sessions, such as a user's login information or settings for a web application.

To use web storage in HTML5, you can use the **localStorage** and **sessionStorage** objects in JavaScript. Here's an example of how to store and retrieve data using local storage:

```
    // store data in local storage
localStorage.setItem("username", "john");

// retrieve data from local storage
var username = localStorage.getItem("username");
console.log(username); // outputs "john"
```

In this example, we use the **setItem()** method to store

the key-value pair "username" and "john" in local storage. We then use the **getItem()** method to retrieve the value of the "username" key and store it in a variable.

Web storage provides a convenient way to store data on the client-side and can be used to create more responsive and interactive web applications. However, it's important to be mindful of the amount of data being stored and to always consider the privacy implications of storing data on a user's device.

## Understanding Geolocation

Geolocation is a feature of modern web browsers that allows web developers to retrieve the geographical location of a user's device. This can be useful for a variety of applications, such as providing location-based services, tracking the movements of a user, or customizing the content of a web application based on the user's location.

To use geolocation in HTML5, you can use the **navigator.geolocation** object in JavaScript. Here's an example of how to retrieve the user's current location:

```
// retrieve the user's location
navigator.geolocation.getCurrentPosition(function(position) {
  var latitude = position.coords.latitude;
  var longitude = position.coords.longitude;
  console.log("Latitude: " + latitude + ", Longitude: " + longitude);
});
```

In this example, we use the **getCurrentPosition()** method to retrieve the user's current position. The method takes a callback function as an argument, which is called with a **Position** object containing the latitude and longitude coordinates of the user's location.

It's important to note that geolocation is a sensitive feature that requires the user's permission to access. Most modern web browsers will prompt the user for permission before allowing a web application to access their location. As a web developer, it's important to be transparent about how you're using a user's location data and to respect their privacy.

# Understanding Web Workers

Web workers are a feature of modern web browsers that allow web developers to run scripts in the background of a web page without interrupting the user interface. This can be useful for performing computationally intensive tasks or long-running

operations without blocking the main thread of the web page.

Web workers work by running scripts in a separate thread from the main thread of the web page. This separate thread can be used to perform tasks such as data processing, image manipulation, or network requests. When the web worker has completed its task, it sends a message back to the main thread with the result of its computation.

To use web workers in HTML5, you can use the **Worker** object in JavaScript. Here's an example of how to create a web worker:

```
// create a new web worker
var worker = new Worker("worker.js");

// send a message to the web worker
worker.postMessage("hello");

// receive a message from the web worker
worker.onmessage = function(event) {
  console.log("Received message from worker: " + event.data);
};
```

In this example, we create a new web worker by passing the name of a JavaScript file ("worker.js") to the **Worker** constructor. We then send a message to the web worker using the **postMessage()** method and receive a message from the web worker using the **onmessage** event handler.

Web workers provide a powerful way to run scripts in the background of a web page without interrupting the user interface. However, it's important to note that web workers have some limitations, such as not being able to access the DOM or interact with the main thread of the web page. As a web developer, it's important to consider these limitations when deciding whether or not to use web workers in your web application.

## Understanding Drag and Drop

Drag and drop is a user interface feature in HTML5 that allows users to move elements on a web page by clicking and dragging them to a new location. Drag and drop is commonly used for tasks such as rearranging items in a list or moving images on a canvas.

To enable drag and drop in HTML5, you can use the **draggable** attribute on elements that you want to be draggable. Here's an example:

```
<div draggable="true">Drag me!</div>
```

In this example, we've added the **draggable** attribute to a **div** element to make it draggable. When the user clicks and drags the **div**, the browser will display a "ghost" image of the element that the user can drag around the screen.

You can also use the **ondragstart**, **ondrag**, and **ondragend** events to control the behavior of the draggable element. Here's an example:

```
<div draggable="true" ondragstart="dragStart(event)" ondrag="dragging(event)" ondragend="dragEnd(event)">Dra

<script>
function dragStart(event) {
  // do something when the user starts dragging the element
}

function dragging(event) {
  // do something while the user is dragging the element
}

function dragEnd(event) {
  // do something when the user stops dragging the element
}
</script>
```

In this example, we've added event handlers for the **ondragstart**, **ondrag**, and **ondragend** events to the **div** element. These event handlers allow us to control the behavior of the draggable element, such as by changing the appearance of the "ghost" image or updating the position of the element as the user drags it around the screen.

Drag and drop is a powerful user interface feature that can greatly enhance the interactivity of web applications. However, it's important to ensure that drag and drop is implemented in a way that is accessible and usable for all users, including those who may have difficulty using a mouse or touch screen.

# Understanding Web Sockets

Web Sockets is a protocol in HTML5 that enables two-way communication between a client and a server

over a single, long-lived connection. This allows for real-time, event-driven communication between the client and server, and is ideal for applications such as online gaming, chat applications, and financial trading platforms.

Web Sockets use the WebSocket API, which is built into modern web browsers and provides a simple interface for creating and managing WebSocket connections. Here's an example of how to create a WebSocket connection in JavaScript:

```
<script>
var socket = new WebSocket("ws://example.com/socket");

socket.onopen = function(event) {
  // do something when the connection is opened
}

socket.onmessage = function(event) {
  // handle incoming messages from the server
}

socket.onclose = function(event) {
  // do something when the connection is closed
}

</script>
```

In this example, we create a new WebSocket connection to a server at **ws://example.com/socket**. We then define event handlers for the **onopen**, **onmessage**, and **onclose** events, which are triggered when the connection is opened, when incoming messages are received from the server, and when the connection is closed, respectively.

Once the WebSocket connection is established, you can use the **send()** method to send messages to the server, and the server can use the same WebSocket connection to send messages back to the client. This

allows for real-time, bidirectional communication between the client and server.

## Accessibility and SEO

Accessibility and SEO are two important factors to consider when developing websites with HTML. Accessibility refers to designing and developing websites that are accessible to all users, including those with disabilities or impairments. SEO, or search engine optimization, refers to the process of optimizing a website's content and structure to improve its visibility and ranking in search engine results pages (SERPs).

Here are some tips for improving accessibility and SEO in HTML:

**Accessibility**:

- Use semantic HTML elements, such as headings, lists, and tables, to structure content in a meaningful and accessible way.
- Provide alternative text descriptions for images, videos, and other non-text content, using the "alt" attribute in HTML.
- Use descriptive link text that accurately reflects the destination of the link.
- Ensure that color is not the only means of conveying information, as users with visual impairments may not be able to distinguish between different colors.
- Use keyboard navigation and ensure that all interactive elements can be accessed and activated using only the keyboard.
- Provide captions or transcripts for audio and video content.

**SEO**:

- Use descriptive and relevant titles for pages and headings, using H1, H2, and other heading tags to structure content.
- Use meta descriptions to provide a summary of the content on each page.
- Use descriptive URLs that accurately reflect the content of the page.
- Use keywords and phrases relevant to the content of the page in headings, content, and meta data.
- Ensure that the website is mobile-friendly and responsive, as mobile optimization is a key factor in SEO.
- Use structured data markup to provide additional information to search engines about the content of the page.

# Making your HTML5 Code Accessible

Making HTML5 code accessible means designing and developing web content that can be used by people with disabilities. Accessibility in HTML5 is achieved by using best practices to optimize your code for use with assistive technologies, such as screen readers or braille displays.

Here are some tips for making your HTML5 code accessible:

- **Use semantic HTML elements**: Use headings, paragraphs, lists, and tables to structure content in a meaningful way. These elements help screen readers to interpret the content more accurately.
- **Provide text alternatives**: Use the "alt" attribute to provide text descriptions for images, videos, and other non-text content. This allows users who are visually impaired to understand the content of the page.
- **Use descriptive link text**: Use descriptive text for links, rather than generic text like "click here." This helps users understand the purpose of the link.
- **Use descriptive labels**: Use descriptive labels for form

controls, such as text fields, radio buttons, and checkboxes. This allows screen readers to accurately describe the form to users.

- **Use color carefully**: Do not use color as the only means of conveying information. Ensure that all important information is conveyed through text or other visual cues.
- **Provide keyboard navigation**: Ensure that all interactive elements can be accessed and activated using only the keyboard.
- **Test with assistive technologies**: Test your website using screen readers or other assistive technologies to ensure that it is accessible to users with disabilities.

## Using Semantic Markup

Using semantic markup means using HTML elements to describe the meaning of the content on a web page, rather than using generic elements like "div" and "span" for all content. Semantic markup helps search engines and other automated systems understand the structure and meaning of the content on a web page, which can improve the page's search engine ranking and accessibility for users.

Here are some examples of semantic markup elements and when to use them:

- **<header>**: Use this element to define the header of a web page, including any navigation menus or branding.
- **<nav>:** Use this element to define the navigation links on a web page.
- **<main>:** Use this element to define the main content of a web page.
- **<article>:** Use this element to define a self-contained unit of content, such as a blog post or news article.
- **<section>:** Use this element to define a section of content that is thematically related, such as a chapter in a book or a section of a news article.
- **<aside>:** Use this element to define content that is related to the main content, but not central to it, such as a sidebar

or advertisement.

- **<footer>:** Use this element to define the footer of a web page, including any copyright information or contact details.

## Applying Semantic Markup

To apply semantic markup, you need to choose the appropriate HTML element that best describes the meaning of the content you want to display on your web page. Here are some examples of how to apply semantic markup:

1. Use <header> for the header section of your web page.

```
<header>
    <h1>My Website</h1>
    <nav>
      <ul>
        <li><a href="#">Home</a></li>
        <li><a href="#">About</a></li>
        <li><a href="#">Contact</a></li>
      </ul>
    </nav>
  </header>
```

1. Use <main> for the main content area of your web page.

```html
<main>
    <article>
      <h2>Article Title</h2>
      <p>Article content goes here.</p>
    </article>
    <aside>
      <h3>Related Articles</h3>
      <ul>
        <li><a href="#">Related Article 1</a></li>
        <li><a href="#">Related Article 2</a></li>
        <li><a href="#">Related Article 3</a></li>
      </ul>
    </aside>
  </main>
```

1. Use <nav> for the navigation links on your web page.

```html
<nav>
    <ul>
      <li><a href="#">Home</a></li>
      <li><a href="#">About</a></li>
      <li><a href="#">Contact</a></li>
    </ul>
  </nav>
```

1. Use <footer> for the footer section of your web page.

```html
    <footer>
  <p>&copy; 2023 My Website</p>
</footer>
```

1. Use <section> to group related content on your web page.

```
<section>
    <h2>Section Title</h2>
    <p>Section content goes here.</p>
</section>
```

## Optimizing for Search Engines

Here are some tips for optimizing your HTML5 code for search engines:

- **Use descriptive page titles and meta descriptions**: Use descriptive page titles and meta descriptions to give search engines and users a clear idea of what your page is about. Include relevant keywords, but don't stuff them unnaturally.

```
<head>
    <title>My Website - Home</title>
    <meta name="description" content="A description of my website">
</head>
```

- **Use header tags appropriately**: Use header tags (**h1**, **h2**, **h3**, etc.) to structure your content in a logical hierarchy. This helps search engines understand the importance of different sections of your page.
- Include relevant keywords in your content: Use relevant keywords in your content, but don't overdo it. Make sure the keywords fit naturally into the content.
- **Use descriptive and relevant URLs**: Use descriptive and relevant URLs that reflect the content of your page. Avoid using long or irrelevant URLs.

```
<a href="/blog/article-name">Read the article</a>
```

- **Optimize your images**: Optimize your images by using descriptive file names and alt text, and compressing the file

size for faster loading times. This can help your images show up in image searches.

```
<img src="example.jpg" alt="A description of the image">
```

- **Use internal linking:** Use internal links to connect related pages and content on your website. This helps search engines understand the organization of your website and the relationship between different pages.

```
<a href="/blog">Visit our blog</a>
```

CHAPTER SIX

# HTML PRACTICAL EXERCISES

The HTML Practical Exercises chapter is a collection of practical exercises designed to help you master HTML coding skills by creating various types of websites. This chapter includes a range of projects, from simple to complex, to suit the needs of different skill levels. These projects cover a wide range of topics, including basic HTML tags and attributes, forms, tables and lists, multimedia, canvas, and SVG graphics, and advanced HTML features like web storage, geolocation, and web workers. Each project is designed to challenge your coding abilities and help you develop your problem-solving skills. By completing these projects, you can gain a better understanding of how HTML works, improve your code quality, and create functional and visually appealing websites. Whether you are a beginner or an experienced web developer, the HTML Practical Exercises chapter can help you refine your skills and become a proficient HTML coder.

## Exercise 1: Creating a Login Page

You will need to first create a folder for your code files, ensuring that both the index file and the subordinate files are together in the same folder. Do this for every single project you are carrying out.

```
<!DOCTYPE html>

<html>

<head>

    <meta charset='utf-8'>

    <meta http-equiv='X-UA-Compatible' content='IE=edge'>

    <title>Example Page</title>

    <meta name='viewport' content='width=device-width, initial-scale=1'>

    <link rel='stylesheet' type='text/css' media='screen' href='main.css'>

    <script src='main.js'></script>

</head>

<body>

    <form>

        <h1>Login Page</h1>

        <label for="username">Username:</label>

        <input type="text" id="username" name="username" required>

        <label for="password">Password:</label>
```

*<input type="password" id="password" name="password" required>*

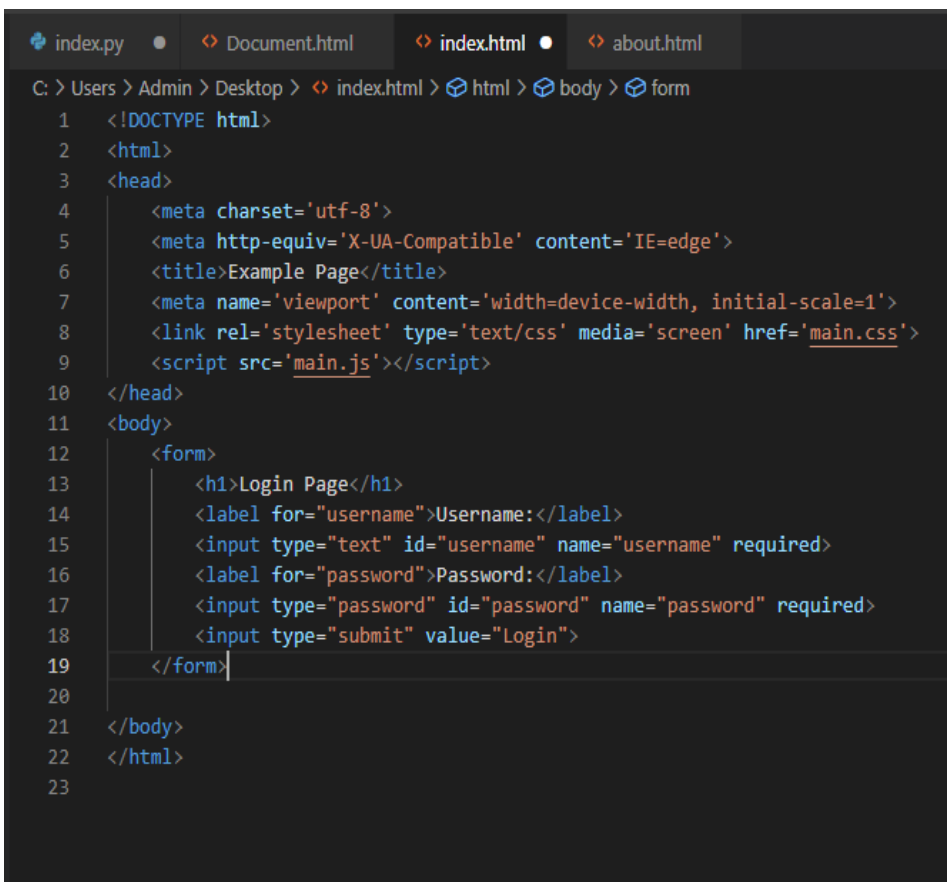
*<input type="submit" value="Login">*

*</form>*

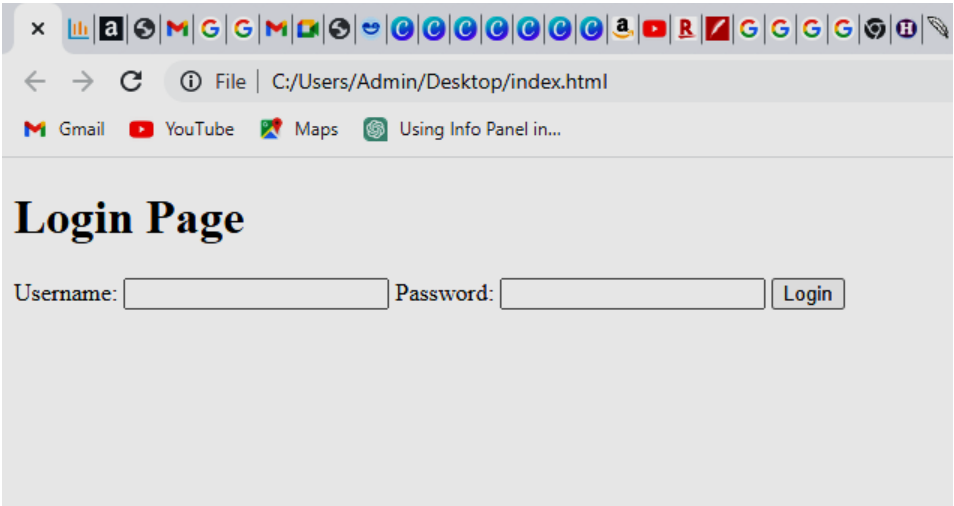
*</body>*

*</html>*

Create an HTML file with the name “**index.html**” and replicate the code above.



See the result below.




## Exercise 2: Creating a Registration Page

Below is an example of a basic registration form using HTML code:

```
<!DOCTYPE html>

<html>

<head>

    <meta charset='utf-8'>

        <meta http-equiv='X-UA-Compatible' content='IE=edge'>

    <title>Example Page</title>

        <meta name='viewport' content='width=device-width, initial-scale=1'>
```

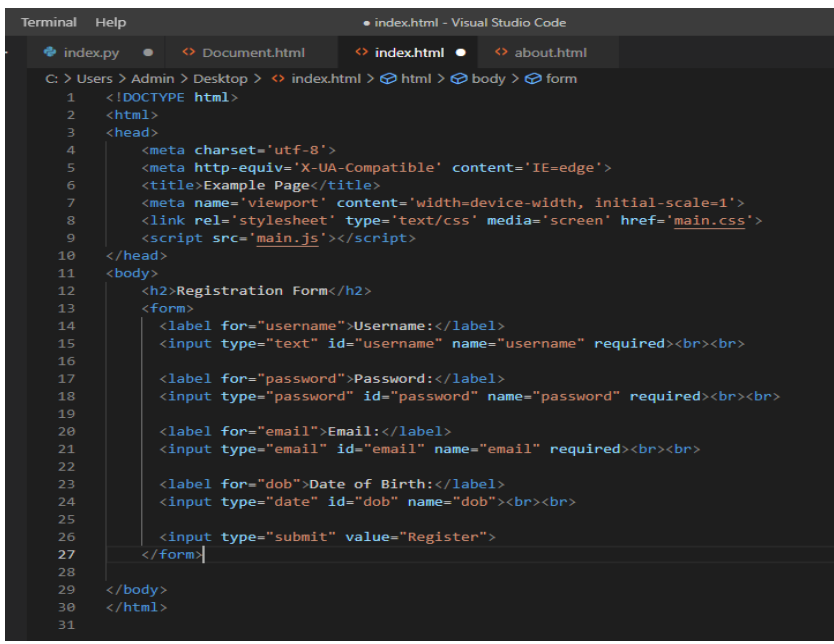
```
    <link rel='stylesheet' type='text/css' media='screen' href='main.css'>

    <script src='main.js'></script>

</head>

<body>

    <h2>Registration Form</h2>

    <form>

        <label for="username">Username:</label>

                    <input type="text" id="username" name="username" required><br><br>

        <label for="password">Password:</label>

                <input type="password" id="password" name="password" required><br><br>

        <label for="email">Email:</label>

            <input type="email" id="email" name="email" required><br><br>

        <label for="dob">Date of Birth:</label>

                        <input type="date" id="dob" name="dob"><br><br>

        <input type="submit" value="Register">
```

*</form>*

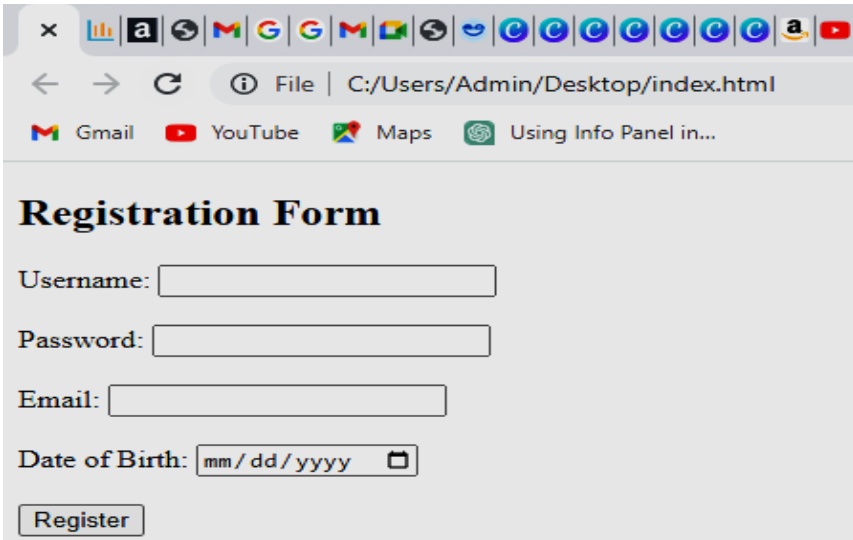
*</body>*

*</html>*

The representation of the above code in Visual Studio Code is shown below.




See the result below.




## Exercise 3: Create a Simple Personal Portfolio Website

It should include the following:

- **Home page**: This page should include your name, a brief introduction about yourself, and links to your other pages.
- **About page**: This page should provide more detailed information about yourself, your skills, education, and work experience.
- **Projects page**: This page should showcase some of your past projects and include descriptions and images.
- **Contact page**: This page should include a contact form where visitors can reach out to you, as well as your social media links and any other relevant contact information.

The code below will guide you on this.

```
<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">
```

```
        <meta http-equiv="X-UA-Compatible" content="IE=edge">

        <meta name="viewport" content="width=device-width, initial-scale=1.0">

        <title>Project 3</title>

    </head>

    <body>

        <body>

        <header>

            <h1>Sammie Smith</h1>

            <nav>

                <ul>

                    <li><a href="index.html">Home</a></li>

                    <li><a href="about.html">About</a></li>

                    <li><a href="projects.html">Projects</a></li>

                    <li><a href="contact.html">Contact</a></li>

                </ul>

            </nav>

        </header>
```

<main>

<section>

<h2>About Me</h2>

A certified Computer Scientist, HSE professional, writer, front-end developer using (HTML, CSS, and JavaScript), can run basic programs using (Java, C++, and Visual Basic), a graphic designer, proficient in the use of Microsoft Office Suite, an accomplished teacher, and a counselor, instructing classes of different sizes and ability levels, experienced in business sales, experienced in agro extension services, can run software installations and troubleshoot basic computer problems. An analytical thinker, self-motivated, results-driven professional with experience in leadership, organization, finance management, and people management, a fast learner that pays close attention to details and is ready to learn something new. I seek to deploy my skills and my knowledge to promote the growth and advancement of the organization and pursue her goals to realize the objectives of the organization while building a strong career for myself.

</section>

<section>

*<h2>My Skills</h2>*

*<ul>*

*<li>Writing</li>*

*<li>Menial skill in the use of HTML,CSS, JAVA, Visual Basic and C++ Programming Languages</li>*

*<li>Proficient in Microsoft Word, Excel, Power Point Publisher, Access etc.</li>*

*<li>Fast learner, good team player, result oriented with excellent motivational skills </li>*

*<li>Advance skill in the use of Corel Draw and Photoshop</li>*

*</ul>*

*</section>*

*<section>*

*<h2>Education</h2>*

*<p>*

* 2018-2019: The Federal Polytechnic, Bida<br>*

*Higher National Diploma (Upper Credit) – Computer Science.<br>*

*2015-2016: Niger State Polytechnic, Zungeru<br>*

*National Diploma (Distinction) - Computer Science.<br>*

*2013: Day Secondary School, Eyagi Bida<br>*

*SSCE (O'Level).<br>*

*2011-2013: Government Technical College, Bida<br>*

*SSCE (O'Level).<br>*

*2007-2010: Day Secondary School, Gaba, Niger State <br>*

*JSCE <br>*

*2000-2006: St. Jude's Nursery/Primary School Gaba, Niger State <br>*

*First School Leaving Certificate.<br>*

* *

*</section>*

*<section>*

*<h2>Work Experience</h2>*

* *

* National Youth Service Corps: 18-05-2021 – 17-05-2022.<br>*

*Taught VOC1, VOC2 & VOC3 students Computer Craft Studies (CSS), make lesson notes, organized practical classes for the students using a projector, assess the student's performances during practical sections, conducted exams for the students, run end*

*of term school projects for the students, scanned, type and print documents.*

*<br>Jiwo Academy Dutsen Kura Gwari, Minna: 10-02-2021 – 02-05-2021*

*<br> I taught the Secondary school section as an ICT subject Teacher, and organized computer practical classes for the students, I also volunteered to type the student's test questions, input the students' scores, and generate their exam report sheet using a computer program and afterward printed them.*

*POS Partnership Business: 11-06-2020 – 20th-04-2021<br>*

*Transaction management, payment processing, sales reporting, customer relationship management, and employee management.</ b><br>*

*<br>*

*N-power: 04-08-2018 – 30-07-2020 <br>*

*Served as Agro Extension Officer under Lavun LGA (Gaba Ward), took farmer's Data and Survey using ODK COLLECT App, sensitizing farmers on good farm practices.*

*<br>*

*St. Jude's Nursery/Primary School, Gaba: 01-09-2013 – 05-01-2015<br>*

*Taught the pupils of Primary Six (6) as a Class teacher taking them all subjects, taking register attendance, making lesson notes, and writing lesson plans. The school that almost closed down witnessed a great*

*turnaround academically and financially as they yield to my advice on school management.*

*<br>Recharge Card Retailer: 01-02-2010 – 05-01-2011<br>*

*<br>  Buying recharge cards from wholesalers, selling the recharge card to the customers, and maximizing profits.<br>*

*<br>*

*Volunteer Experience:<br>*

*<br>  Helped to upload GTC teachers' E-affidavit and other documents on their Rivers State teacher's dashboard, from 31st August to 11th September 2021.*

*<br> Helped to design Divine Grace POS Berners, also helped to design jotter cover page for NCCF Opobo Zone for their batch B 2020 NCCF Send forth.*

* *

*</section>*

*</main>*

*<footer>*

* Contact Me: *

*<ul>*

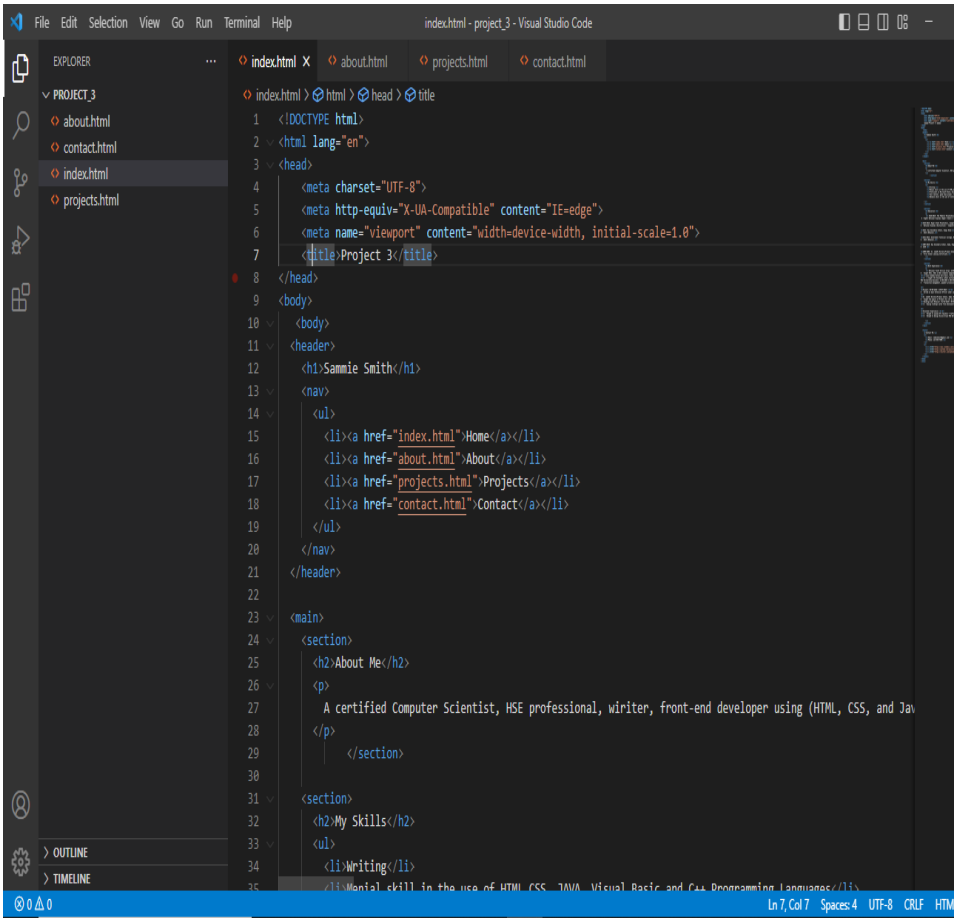
*<li>Email: sammiesmith@gmail.com</li>*

*<li>Phone: 123-456-7890</li>*

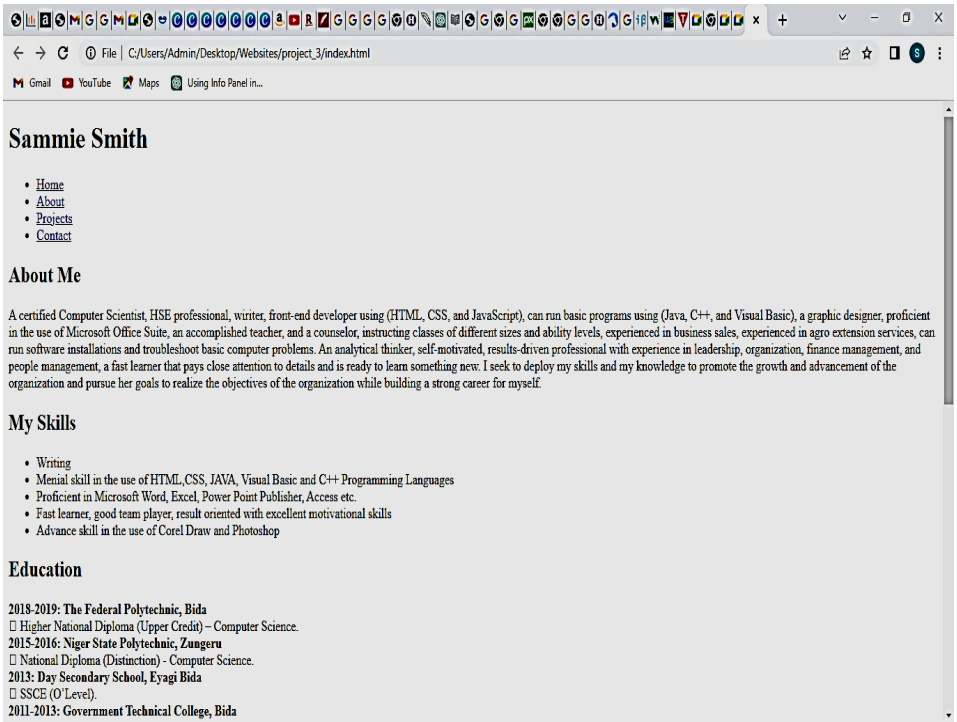
```
    </ul>

    <ul>

        <li><a  href="https://www.linkedin.com/in/myprofile/">LinkedIn</a></
li>

      <li><a href="https://github.com/myusername">GitHub</a></li>

      <li><a href="https://twitter.com/myhandle">Twitter</a></li>

    </ul>

  </footer>

</body>

</html>
```

You can create a new file for About and save it with “about.html”, Projects and save with

“projects.html”, Contact and save with “contact.html” within the same folder. Copy and paste the index file code on each of the files created, then remove the necessary elements that are not needed, and replace them with the required information. When that is done, your project would be completed. The screenshot below contains part of the code.



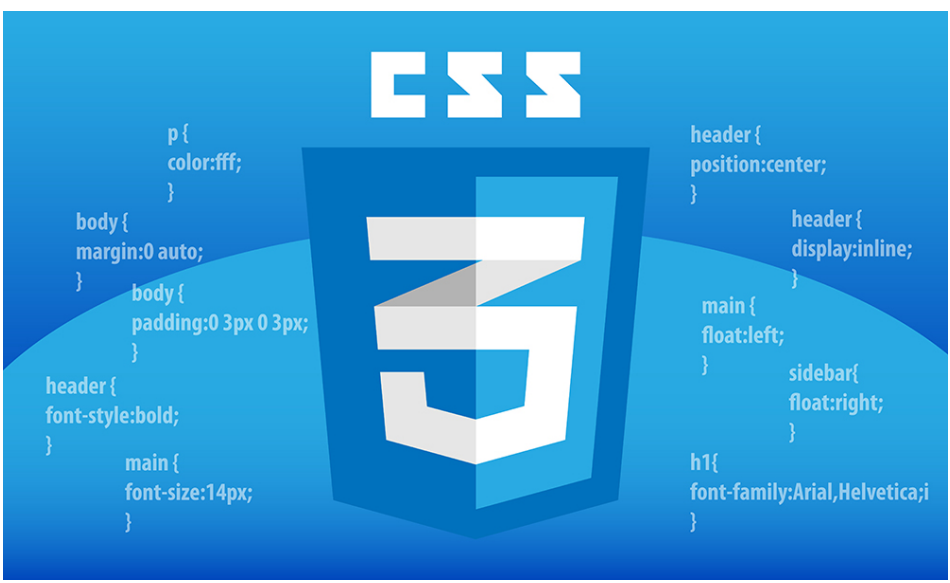

See the result from the browser below.


File | C:/Users/Admin/Desktop/Websites/project_3/index.html
Gmail
YouTube
Maps
Using Info Panel in...
Sammie Smith
Home
About
Projects
Contact
About Me
A certified Computer Scientist, HSE professional, wiriter, front-end developer using (HTML, CSS, and JavaScript), can run basic programs using (Java, C++, and Visual Basic), a graphic designer, proficient in the use of Microsoft Office Suite, an accomplished teacher, and a counselor, instructing classes of different sizes and ability levels, experienced in business sales, experienced in agro extension services, can run software installations and troubleshoot basic computer problems. An analytical thinker, self-motivated, results-driven professional with experience in leadership, organization, finance management, and people management, a fast learner that pays close attention to details and is ready to learn something new. I seek to deploy my skills and my knowledge to promote the growth and advancement of the organization and pursue her goals to realize the objectives of the organization while building a strong career for myself.
My Skills
Writing
Menial skill in the use of HTML,CSS, JAVA, Visual Basic and C++ Programming Languages
Proficient in Microsoft Word, Excel, Power Point Publisher, Access etc.
Fast learner, good team player, result oriented with excellent motivational skills
Advance skill in the use of Corel Draw and Photoshop
Education
2018-2019: The Federal Polytechnic, Bida
 Higher National Diploma (Upper Credit) – Computer Science.
2015-2016: Niger State Polytechnic, Zungeru
 National Diploma (Distinction) - Computer Science.
2013: Day Secondary School, Eyagi Bida
 SSCE (O'Level).
2011-2013: Government Technical College, Bida

CHAPTER SEVEN

# INTRODUCTION TO CSS

CSS (Cascading Style Sheets) is a language used to describe the presentation of a document written in HTML (Hypertext Markup Language). It is used to add style and formatting to web pages and can control the layout, typography, colors, and other visual elements of a webpage. CSS can be used to create responsive designs that adapt to different screen sizes and devices, and it can be used in conjunction with HTML and JavaScript to create dynamic and interactive web pages. Understanding CSS is essential for creating professional-looking and functional websites.




# What is CSS?

CSS stands for Cascading Style Sheets. It is a styling

language used to describe the presentation of a document written in HTML or XML, including colors, layout, fonts, and other visual elements. CSS separates the content of a web page from its presentation, allowing for more flexible and efficient website design. By using CSS, web developers can control the look and feel of a website across multiple web pages and devices, making it easier to maintain and update.

# How is CSS used in Web Development?

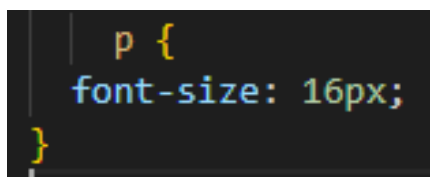
CSS works by selecting HTML elements and defining how they should be displayed. For example, you can select all paragraphs on a page and set their font size, color, and line height to create a consistent look throughout the document. CSS can be applied to HTML elements directly or through classes and IDs, which allows for more targeted styling.

CSS is an essential part of modern web development and is used to create visually appealing and user-friendly websites. It allows developers to separate the content and structure of a webpage from its presentation, making it easier to maintain and update. With CSS, you can create responsive designs that adapt to different screen sizes and devices, and add animations and interactive elements to enhance the user experience.

# How CSS works with HTML

CSS works with HTML by targeting and styling the HTML elements within a webpage. HTML provides the structure and content of the webpage, while CSS is used to control the presentation and layout of that content.

CSS selectors are used to target HTML elements, and then properties are applied to those elements to change their appearance. For example, the following CSS code would target all <p> elements and set their font size to 16 pixels:




This would make all paragraphs within the HTML document have a font size of 16 pixels. CSS can be applied to HTML documents in a few different ways, such as through an external stylesheet file or inline styles within the HTML document.

# Understand the Difference Between CSS and HTML

HTML and CSS are two separate languages used in web development. HTML, or HyperText Markup Language, is used for creating the structure and content of a web page. It defines the various elements that make up the page, such as headings, paragraphs, images, and links.

CSS, or Cascading Style Sheets, is used for styling the presentation of the web page. It defines how the HTML elements should be displayed, such as the color, font, size, and layout. CSS can be used to create complex page layouts and visually appealing designs.

In short, while HTML defines the content and structure of a web page, CSS defines its presentation and style. They work together to create a complete and functional website.

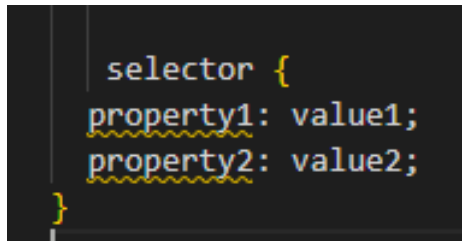
# Benefits of using CSS

There are several benefits of using CSS in web development:

- **Separation of presentation and content**: CSS allows for a clear separation between the presentation and content of a web page, making it easier to maintain and update the design without affecting the content.
- **Consistency**: CSS enables the designer to apply consistent styles to multiple pages, ensuring that the look and feel of the website are uniform.
- **Flexibility**: CSS provides flexibility in design by allowing for the use of various layout and positioning techniques, as well as the ability to create responsive designs that adapt to different screen sizes.
- **Faster page load times**: By separating the presentation and content, CSS reduces the amount of code needed to load a web page, resulting in faster page load times.
- **Accessibility**: CSS makes it easier to create accessible websites by providing the ability to apply styles to specific elements, such as headings and links, and improving the readability of the content.

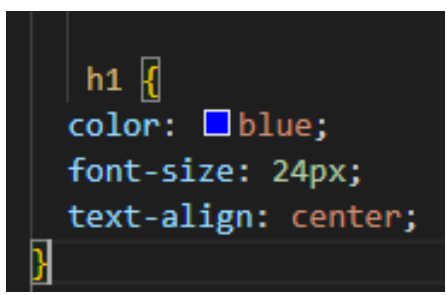
# Basic CSS Syntax

The basic syntax of CSS consists of a selector and a set of declarations:




The selector selects the HTML element(s) that you want to apply the styles, and the declarations define the styles themselves. Each declaration consists of a property and its value, separated by a colon.

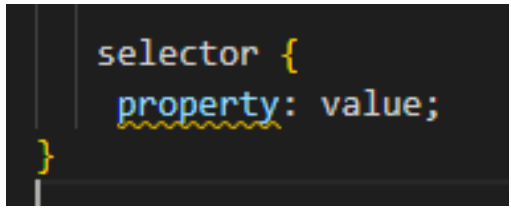
Here is an example:




In this example, **h1** is the selector that targets all **h1** HTML elements, and the declarations define the styles that will be applied to them: blue text color, 24px font size, and centered text alignment.

# Writing CSS rules

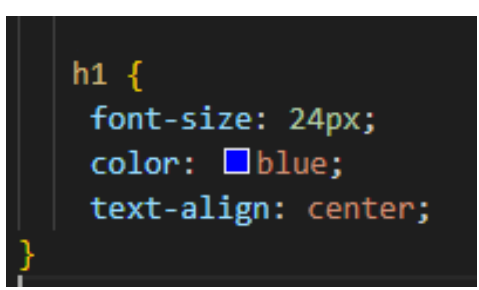
Writing CSS rules involves selecting HTML elements and applying style properties to them. The basic syntax for writing a CSS rule is:




- **Selector**: Specifies which HTML element(s) the rule applies to
- **Property**: Defines the aspect of the element that we want to style
- **Value**: Sets the value of the property

For example, to set the font color of all paragraph elements to red, we would use the following CSS rule:

Multiple properties can be applied to the same selector, by separating them with a semicolon (;). For example:




This rule sets the font size of all h1 elements to 24 pixels, sets the color to blue, and centers the text.

**Selectors and Declarations**

In CSS, selectors and declarations are used to target specific HTML elements and apply styling rules to them.

Selectors are patterns used to select the elements that you want to style, while declarations define the styling rules that you want to apply to those elements. For example, consider the following CSS code:

In this example, the **h1** selector targets all **<h1>** elements on the page, and the declarations inside the curly braces apply the styling rules. The **color** declaration sets the text color to red, and the **font-size** declaration sets the font size to 24 pixels.

Selectors can target specific elements based on their tag name, class, ID, attribute, or a combination of these factors. There are also pseudo-classes and pseudo-elements that allow you to target elements based on their state or position within the document.

Declarations can include a variety of properties, such as color, font, background, border, margin, padding, and more. Each property can have a value assigned to it, such as a color name, a numeric value, a percentage, or a keyword.

# Comments in CSS

In CSS, you can add comments to your code to make it more readable and easier to understand. Comments are not displayed on the web page and do not affect the styling of your elements.

To add comments in CSS, you can use the /* */ syntax. Anything written between /* and */ will be considered a comment and will not be executed by the browser. For example:

```
/* This is a CSS comment */
```

You can also use comments to temporarily disable a block of code without deleting it. This can be useful when testing different styles for your elements. Simply add /* at the beginning of the code you want to disable and */ at the end to re-enable it. For example:

```
/*
h1 {
  color: red;
}
*/

h1 {
  color: blue;
}
```

In the example above, the first block of code is commented out and will not be executed by the browser, while the second block will apply the color

blue to all **h1** elements on the page.

# External and Internal CSS

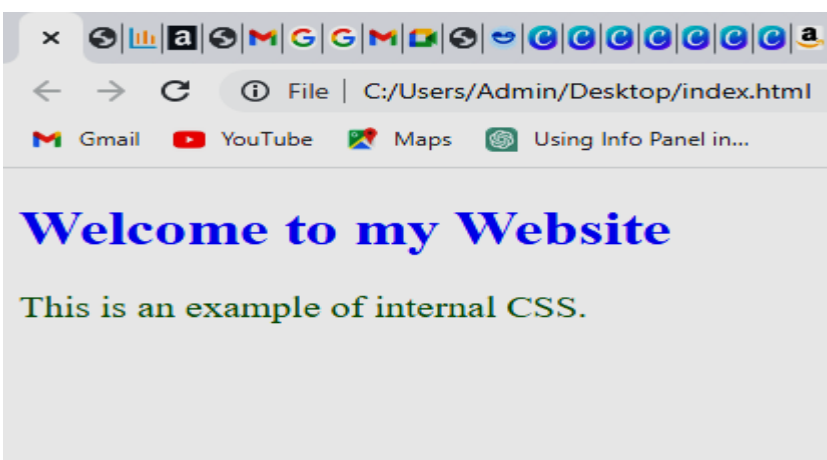
External and Internal CSS are two ways of organizing CSS code on a web page.

Internal CSS is CSS code that is placed within the head section of an HTML document, using the <style> tag. It applies only to the current web page and cannot be used on other pages on the website. Internal CSS is useful for small, single-page websites or for making quick changes to a specific page without affecting the rest of the site.

Here is an example of internal CSS code:

```
Terminal  Help                    index.html - project_3 - Visual Studio Code
index.html C:\...\Desktop ×  index.html .\  about.html  projects.html  contact.ht
C: > Users > Admin > Desktop > index.html > html > body > p
<!DOCTYPE html>
<html>
<head>
    <meta charset='utf-8'>
    <meta http-equiv='X-UA-Compatible' content='IE=edge'>
    <title>Example Page</title>
    <meta name='viewport' content='width=device-width, initial-scale=1'>
    <link rel='stylesheet' type='text/css' media='screen' href='main.css'>
    <script src='main.js'></script>

     <style>
      h1 {
        color: blue;
        font-size: 32px;
      }
      p {
        color: green;
        font-size: 20px;
      }
    </style>
  </head>
  <body>
    <h1>Welcome to my Website</h1>
    <p>This is an example of internal CSS.</p>
</body>
</html>
```

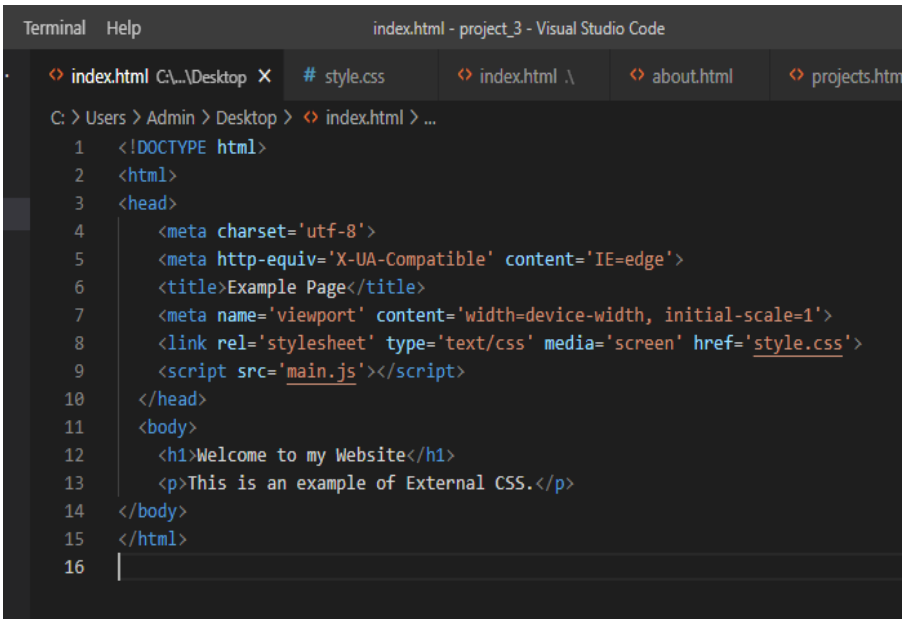
See the corresponding result below.




External CSS is CSS code that is stored in a separate .css file and linked to the HTML document using the <link> tag in the head section of the HTML document.

This allows the CSS to be used across multiple web pages on a website. External CSS is useful for larger websites with multiple pages and for maintaining consistency across the site.

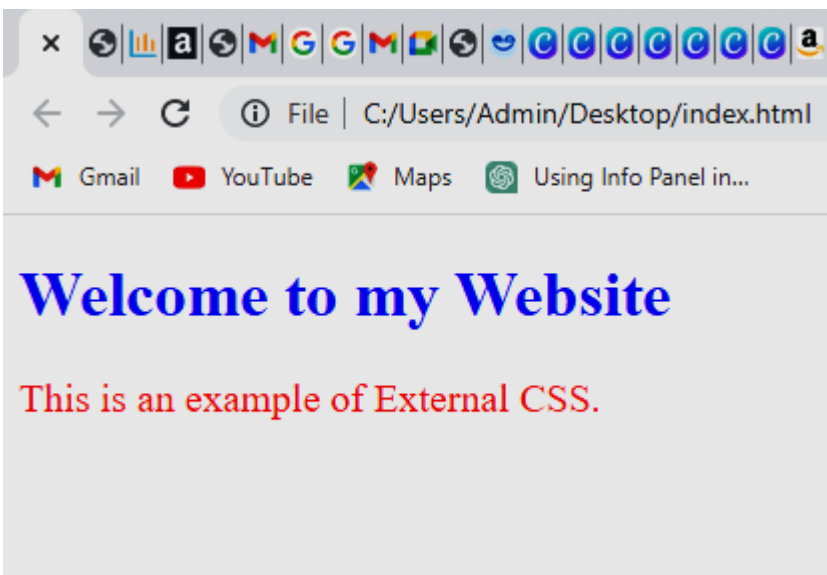
Here is an example of external CSS code:




Below is the corresponding style.css file:

```
Terminal  Help                                        style.css

<> index.html C:\...\Desktop        # style.css  ×

C: > Users > Admin > Desktop > # style.css > ...
1   h1 {
2       color: blue;
3       font-size: 32px;
4   }
5   p {
6       color: red;
7       font-size: 20px;
8   }
9
```

The corresponding result is shown below.




# Linking an External CSS

Linking an external CSS file to an HTML document involves the following steps:

- Create a CSS file: Create a new file and save it with a

".css" extension, e.g. "styles.css". This file will contain all of your CSS code.

- Add CSS code: Add your CSS code to the "styles.css" file. This can include any selectors, declarations, and rules that you want to apply to your HTML document.
- Link the CSS file to your HTML document: In the head section of your HTML document, add a link element that specifies the location of the CSS file. The link element should look like this:

```
<link rel='stylesheet' type='text/css' media='screen' href='style.css'>
<script src='main.js'></script>
```

- You can replace the "styles.css" name or path with the actual path to your CSS file. This path can be a relative or absolute URL.

- Save your files: Save both your HTML and CSS files.

- After these steps, your HTML document will be linked to the external CSS file, and any styles you defined in the CSS file will be applied to your HTML elements.

CHAPTER EIGHT

# CSS STYLE PROPERTIES

CSS Style Properties are essential tools for web developers to customize the appearance of HTML elements. These properties allow developers to control the size, color, font, layout, and other visual characteristics of web pages. In this chapter, we will explore the various CSS Style Properties available and how they can be used to create visually appealing websites, alongside several other CSS components. We will also look at how to combine multiple style properties to create complex designs and layouts. By the end of this chapter, you will have a good understanding of how to use CSS Style Properties to customize the look and feel of your web pages.

## Text Properties

CSS Text Properties refer to the different properties that are used to style text content in HTML documents. These properties include font styles, font sizes, font weights, line heights, text alignments, and text decorations. With CSS Text Properties, you can control the appearance of text on your website, making it more visually appealing and easy to read.

In this chapter, we will explore the different CSS Text Properties and how they can be used to enhance the appearance of your website's text content. We will also provide examples and best practices for using these properties effectively, so you can create beautiful and

readable text on your website. Whether you are a beginner or an experienced web developer, understanding CSS Text Properties is essential for creating great web content.

# What are CSS Text Properties?

Text properties in CSS are used to style and format the content of text elements within an HTML document. These properties allow developers to control the font family, font size, font weight, line height, text alignment, text decoration, and text transformation.

By using text properties in CSS, you can enhance the readability and appearance of your web pages. For example, you can change the font size to make the text more legible, adjust the line height to improve the spacing between lines, and add text decoration to highlight important text.

Some of the commonly used text properties in CSS include font-family, font-size, font-weight, line-height, text-align, text-decoration, and text-transform. In the following sections, we will explore these properties in more detail and learn how to use them to create effective and attractive web designs.

#### Applying Text Properties

To apply text properties in CSS, you can use the following syntax:

```
selector {
  property1: value1;
  property2: value2;
  ...
}
```

Here's an example of applying text properties to an **h1** element:

```
h1 {
  font-family: Arial, sans-serif;
  font-size: 32px;
  font-weight: bold;
  color: #333;
  text-align: center;
  text-transform: uppercase;
  line-height: 1.2;
  letter-spacing: 1px;
}
```

In this example, we set the font family to Arial or any sans-serif font, font size to 32 pixels, font-weight to bold, color to #333, text alignment to center, text transformation to uppercase, line height to 1.2 times the font size, and letter spacing to 1 pixel.

These text properties can be applied to any HTML element, such as paragraphs, headings, and lists, to change their appearance and improve the readability of the text.

**Font Properties**

Font properties in CSS are used to control the size, style, weight, and family of the text. Here are some common font properties:

- **font-size**: sets the size of the font. It can be specified in pixels, ems, rems, or other units.
- **font-style**: sets the style of the font, such as italic, oblique, or normal.
- **font-weight**: sets the weight of the font, such as bold, bolder, lighter, or normal.
- **font-family**: sets the font family for the text, such as Arial, Times New Roman, or Verdana.
- **text-transform**: transforms the text to uppercase, lowercase, or capitalized.
- **text-decoration**: adds decoration to the text, such as underline, overline, line-through, or none.

**Applying Font Properties**

To apply these font properties, you can use CSS rules with selectors to target specific HTML elements, and then add the desired font properties to the declarations within the rules. For example:

```css
h1 {
  font-size: 36px;
  font-weight: bold;
  font-family: Arial, sans-serif;
  text-transform: uppercase;
  text-decoration: underline;
}
```

This rule applies to all **<h1>** elements and sets the font size to 36 pixels, the font weight to bold, the font family to Arial or sans-serif (if Arial is not available), the text-transform to uppercase, and the text-decoration to underline.

**Color Properties**

Color properties in CSS allow you to specify the color of text, background, borders, and other elements on a webpage. There are several ways to define colors in CSS, including:

- **Named colors:** CSS has predefined names for 140 different colors, such as "red", "blue", "green", etc.
- **Hexadecimal notation**: This method allows you to specify a color using a six-digit code that represents the amount of red, green, and blue in the color.
- **RGB notation**: This method specifies the amount of red, green, and blue in a color using three numbers between 0 and 255.
- **HSL notation**: This method specifies the color using its hue, saturation, and lightness values.

In addition to specifying colors for specific elements, you can also use CSS to apply color to specific parts of an element, such as its border or background. CSS also allows you to apply transparency to colors using the "opacity" property.

# Applying Color properties

To apply color properties in CSS, you can use different color models like RGB, RGBA, Hexadecimal, HSL, HSLA, etc. Here are some examples:

1. Using RGB values:

```
color: rgb(255, 0, 0); /* red */
background-color: rgb(255, 255, 0); /* yellow */
```

1. Using Hexadecimal values:

```
color: #FF0000; /* red */
background-color: #FFFF00; /* yellow */
```

1. Using HSL values:

```
color: hsl(0, 100%, 50%); /* red */
background-color: hsl(60, 100%, 50%); /* yellow */
```

You can also apply color to specific elements using selectors:

```
h1 {
  color: #FF0000;
}

p {
  color: rgb(0, 128, 0);
}
```

You can also set the opacity of colors using RGBA and HSLA values:

```
  color: rgba(255, 0, 0, 0.5); /* semi-transparent red */
background-color: hsla(60, 100%, 50%, 0.8); /* semi-transparent yellow */
```

In addition to text color and background color, you can also use color properties to set border color, box shadow color, and other visual effects in CSS.

## Background Properties

related attributes of an element. These properties allow designers to customize the background of a webpage or specific sections of a webpage to make it more visually appealing.

The background properties in CSS include:

- **background-color**: This property is used to set the background color of an element.
- **background-image**: This property is used to set the background image of an element.
- **background-repeat**: This property is used to specify whether or not the background image should be repeated vertically, horizontally, or not at all.
- **background-position**: This property is used to set the starting position of the background image.
- **background-size**: This property is used to set the size of the background image.
- **background-attachment**: This property is used to specify whether the background image should be fixed or scroll with the content of the element.

# CSS Box Model

The CSS box model is a layout concept that defines the rectangular boxes that are generated for HTML elements in a web page. Each box consists of four parts: content, padding, border, and margin. Understanding the box model is essential for creating well-designed and visually appealing web pages.

The content area is the space where the actual content of an element, such as text or an image, is displayed. The padding is the space between the content and the border. The border is the line that surrounds the content and padding. The margin is the space between the border and the next element on the

page.

The box model is important in CSS because it affects the layout and spacing of elements on a page. By adjusting the padding, border, and margin properties of an element, web developers can control the space around and between elements. This can help create a consistent and visually pleasing layout for a web page.

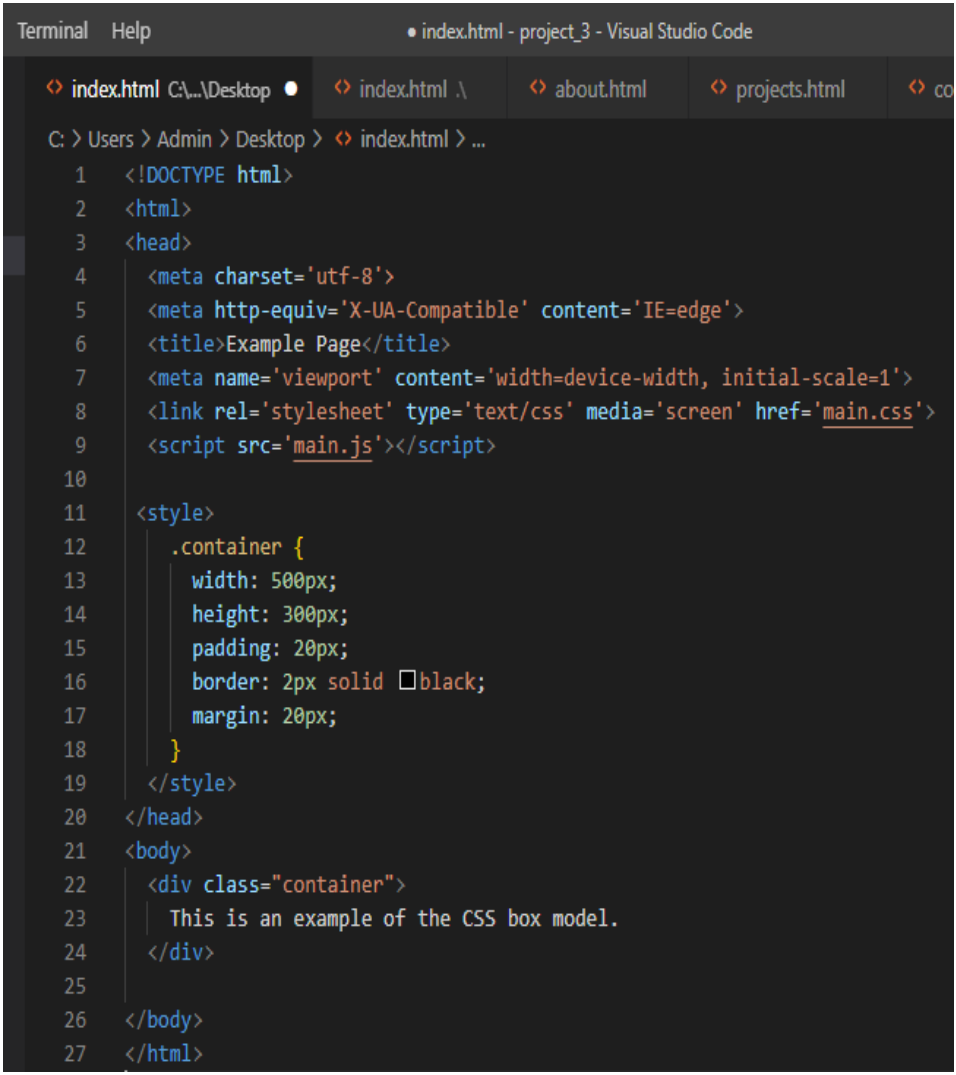
CSS provides several properties for adjusting the box model, including padding, border, and margin. These properties can be set individually for each side of the box, or all sides at once. Additionally, the box-sizing property can be used to control how the width and height of an element are calculated, including whether or not they include padding and border.

# Applying CSS Box Model

To apply the CSS box model, you can use the following properties:

- **Width and height**: You can set the width and height of an element using the **width** and **height** properties. By default, the width and height of an element are set to **auto**.
- **Padding**: You can add padding to an element using the **padding** property. Padding is the space between the element's content and its border.
- **Border**: You can add a border to an element using the **border** property. The border surrounds the element's padding and content.
- **Margin**: You can add a margin to an element using the **margin** property. Margin is the space between the element's border and the adjacent elements.

Here's an example of how to apply the CSS box model:




In this example, we have a **div** element with the class **container**. We have set its width to **500px**, height to **300px**, added **20px** of padding, a **2px** solid black border, and **20px** of margin. The content of the **div** element is "This is an example of the CSS box model."

### Understanding the Box Model

The CSS box model is a fundamental concept in web design that explains how elements are structured on a

web page. Every HTML element is considered a box, and the box model defines how those boxes are laid out, sized, and styled.

The box model consists of four parts: content, padding, border, and margin. The content is the actual text, image, or other media that the element contains. The padding is the space between the content and the border. The border is a line or area that surrounds the padding and content. The margin is the space between the border and the neighboring elements.

The size of each box is determined by the content plus the padding, border, and margin. For example, if an element has a width of 100 pixels and a padding of 10 pixels, the total width of the box would be 120 pixels (100 pixels for the content and 10 pixels of padding on either side).

### Margin, Border, and Padding

Margin, border, and padding are three important properties of the CSS box model. They determine the size and spacing of an element on a web page.

- **Margin**: The space outside the border of an element. It creates a gap between two adjacent elements. You can use a margin to control the amount of white space around an element.
- **Border**: The line that surrounds an element. It can be set to a specific width, style, and color. A border can be used to visually separate an element from other content.
- **Padding**: The space between the content of an element and its border. You can use padding to add space between the content and the border or to create a visual effect within an element.

All of these properties can be set using CSS. You can specify values for each of them separately or use the shorthand property to set them all at once. For example, the following code sets a margin of 10 pixels on all sides, a border with a width of 1 pixel, and a padding of 5 pixels inside the border:

```
.my-element {
  margin: 10px;
  border: 1px solid black;
  padding: 5px;
}
```

**Width and Height**

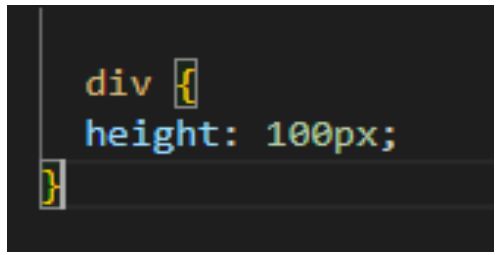
In CSS, the width and height properties are used to set the dimensions of an element.

The **width** property is used to set the width of an element, and it can take a value in pixels, percentages, ems, or other CSS length units. For example:

```
div {
  width: 200px;
}
```

The above CSS rule sets the width of all **div** elements to 200 pixels. The **height** property is used to set the height of an element, and it can also take a value in

pixels, percentages, ems, or other CSS length units. For example:




The above CSS rule sets the height of all **div** elements to 100 pixels. When the **width** and **height** properties are used together with the box model properties (**padding**, **border**, and **margin**), they determine the total size of an element on the web page.

# CSS Layouts

CSS layouts refer to the arrangement of elements on a web page. A good layout is essential for creating an attractive and easy-to-use website. CSS provides a wide variety of tools for creating layouts, including:

- **Display property**: This property determines the type of box an element generates. The most commonly used values are block, inline, and inline-block.
- **Positioning**: This property determines the position of an element on a page. There are four values: static, relative, absolute, and fixed.
- **Float**: This property moves an element to the left or right of its container.
- **Clear**: This property is used to clear floated elements.
- **Flexbox**: This is a new layout module introduced in CSS3. It is a powerful tool for creating flexible and responsive layouts.
- **Grid**: This is another layout module introduced in CSS3. It allows developers to create complex grid-based layouts.

# CSS Floats

Float is a CSS property that allows an element to be positioned to the left or right of its parent container, allowing other content to flow around it. This property is commonly used for creating column layouts and image galleries.

When an element is floated, it is taken out of the normal document flow, meaning that other elements will flow around it. To use the float property, you simply apply it to the element that you want to float, and then set the width and margin properties to position it correctly.

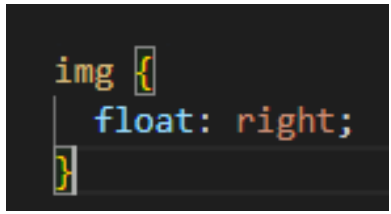
Floats can be useful for creating complex layouts, but they can also cause problems if not used correctly. For example, floated elements may overlap or push other content out of position, and they can also be difficult to work with when it comes to responsive design.

To avoid these issues, it's important to use clearfix techniques to clear floats and ensure that your layout remains stable across different devices and screen sizes. Additionally, CSS Grid and Flexbox layouts provide more modern and flexible alternatives to using floats for layout design.

#### Applying Float

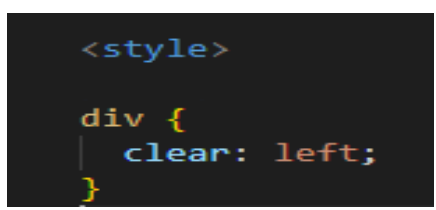
To apply a float in CSS, you need to use the **float** property. The value of the property can be **left** or **right** to specify which direction the element should be floated. For example, if you want to float an image to

the right of a paragraph of text, you would use the following CSS:




This will cause the image to float to the right, allowing text to flow around it on the left.

It's important to note that when you float an element, it will be taken out of the normal flow of the document, which can have an impact on the layout of other elements. To prevent this, you can use the **clear** property to specify that no floating elements should appear on a particular side of the element. For example, to clear all floats to the left of an element, you would use the following CSS:




This will cause the element to be placed below any elements that have been floated to the left.

## Positioning

Positioning in CSS refers to how an element is placed on a web page, either with the browser window, the containing element, or other elements on the page. There are four different types of positioning in CSS:

- **Static Positioning**: This is the default positioning for all HTML elements, where the element appears in the order it is written in the HTML code.
- **Relative Positioning**: This type of positioning allows an element to be moved from its normal position by a specified number of pixels or percentage values, while still taking up its original space in the document flow.
- **Absolute Positioning**: This type of positioning allows an element to be positioned relative to its closest positioned ancestor element, or if none exists, relative to the initial containing block. It is removed from the normal document flow and will not affect the position of other elements on the page.
- **Fixed Positioning**: This type of positioning is similar to absolute positioning, but the element is positioned relative to the browser window rather than its containing element. The element remains in the same position even if the user scrolls the page.

By using CSS positioning, web developers can create complex layouts and positioning effects on their web pages. However, it is important to use positioning judiciously, as it can be difficult to maintain consistency across different screen sizes and devices.

**Applying Positioning**

To apply positioning in CSS, you can use the **position** property. There are four different values you can use for the **position** property:

- **static**: This is the default value, and it means that the element will flow in the normal document flow.
- **relative**: This value positions the element relative to its normal position in the document flow. You can then use the **top**, **bottom**, **left**, and **right** properties to move the element to its normal position.
- **absolute**: This value positions the element relative to its closest positioned ancestor. If there is no positioned ancestor, the element will be positioned relative to the document body. You can then use the **top**, **bottom**, **left**,

and **right** properties to move the element with its closest positioned ancestor.

- **fixed**: This value positions the element relative to the viewport, meaning that it will always stay in the same position even if the page is scrolled. You can then use the **top**, **bottom**, **left**, and **right** properties to move the element with the viewport.

Below is an example of how to use the **position** property:

```css
.box {
  position: relative;
  top: 20px;
  left: 50px;
}
```

This will position the element with the class **box** 20 pixels down and 50 pixels to the right of its normal position in the document flow.

## Display

In CSS, the **display** property is used to define the type of box used to represent an HTML element. It has a variety of values that control how the element is displayed and interacts with other elements on the page.

The most common **display** values are:

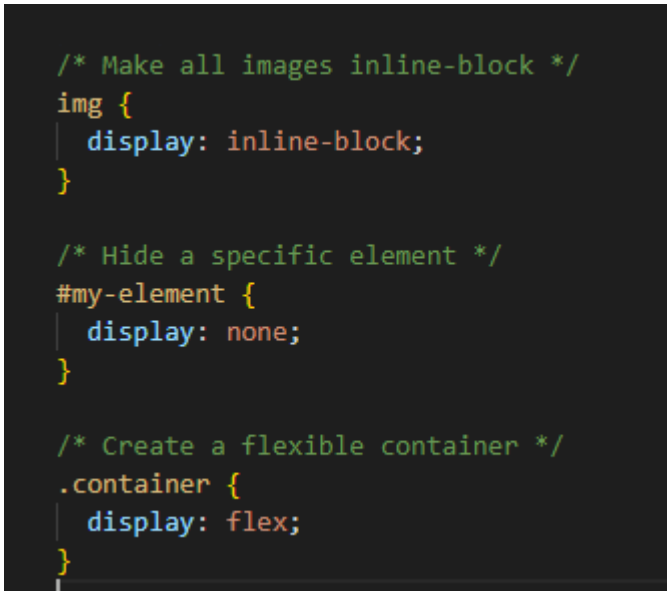
- **block**: The element is displayed as a block-level element, with a line break before and after the element. This is the default display value for elements such as ** ** and **<p>**.
- **inline**: The element is displayed as an inline-level element, with no line break before or after the element. This is the

default display value for elements such as **** and **<a>**.

- **inline-block**: The element is displayed as an inline-level block container, with a line break before and after the element. This is useful for elements that need to be both inline and have a width or height, such as images or form controls.
- **none**: The element is not displayed on the page at all. This is often used for elements that are hidden based on user interaction or for accessibility purposes.
- **flex**: The element is displayed as a flex container, which allows for easy creation of flexible layouts. This is often used for responsive web design.
- **grid**: The element is displayed as a grid container, which allows for more complex layouts than the **flex** property. This is also used for responsive web design.
- **table**: The element is displayed as a table element, with the same behavior as the **<table>** HTML element. This is useful for creating layouts with a fixed number of columns or rows.

**Applying Display**

By using the **display** property, you can control the layout of your web pages and create more effective designs. Below is an example of how to apply the **display** property:



```css
/* Make all images inline-block */
img {
  display: inline-block;
}

/* Hide a specific element */
#my-element {
  display: none;
}

/* Create a flexible container */
.container {
  display: flex;
}
```

# Understanding CSS Grid

CSS Grid is a two-dimensional layout system that allows developers to create complex grid layouts with ease. It is a relatively new addition to CSS, first introduced in 2017, and has quickly become a popular way to create responsive and flexible layouts.

With CSS Grid, developers can define rows and columns and then place content within those areas. This allows for a high degree of flexibility in layout design, as elements can be placed anywhere within the grid, regardless of their order in the HTML document.

CSS Grid also supports responsive design, allowing for easy reordering of content and the ability to change the layout based on screen size or device orientation.

Some key features of CSS Grid include:

- Two-dimensional grid layouts
- Ability to define rows and columns
- Placing content within grid areas
- Support for responsive design
- Control over the sizing and spacing of grid elements

# Applying CSS Grid

To apply CSS Grid, you need to follow these steps:

- Define the parent container as a grid container by setting the display property to the grid.
- Specify the number of rows and columns for the grid using the grid-template-rows and grid-template-columns properties.
- Define the size and spacing of the grid items using the grid-template-rows and grid-template-columns properties.
- Position the grid items within the grid using the grid-row and grid-column properties.

Below is an example of how to apply CSS Grid:

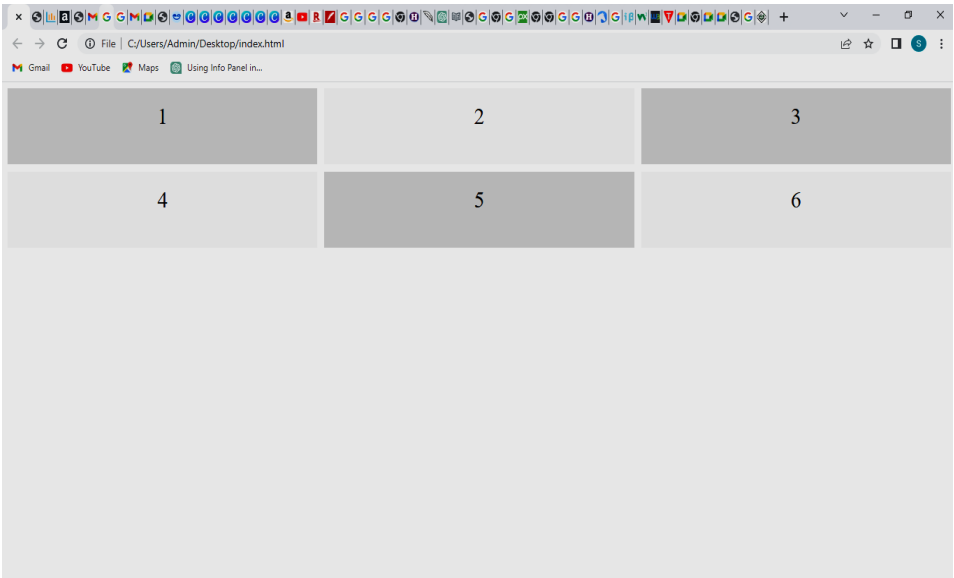
### HTML code:

```
    <div class="grid-container">
  <div class="grid-item">1</div>
  <div class="grid-item">2</div>
  <div class="grid-item">3</div>
  <div class="grid-item">4</div>
  <div class="grid-item">5</div>
  <div class="grid-item">6</div>
</div>
```

### CSS code:

```css
.grid-container {
  display: grid;
  grid-template-columns: repeat(3, 1fr);
  grid-template-rows: repeat(2, 100px);
  grid-gap: 10px;
}

.grid-item {
  background-color: #f2f2f2;
  padding: 20px;
  font-size: 30px;
  text-align: center;
}

.grid-item:nth-child(odd) {
  background-color: #ccc;
}
```

In this example, we have created a grid container with 2 rows and 3 columns using the **grid-template-rows** and **grid-template-columns** properties. We have also set the size and spacing of the grid items using the **grid-template-rows** and **grid-template-columns** properties and positioned them within the grid using the **grid-row** and **grid-column** properties. The result will be a 2x3 grid with 6 grid items inside. The result of the HTML and CSS code above is shown below.




# CSS Animations

CSS Animations refer to the process of adding motion or movement to elements on a web page using CSS (Cascading Style Sheets). This is done by defining specific animations using CSS rules, which can be applied to different HTML elements such as text, images, buttons, and other page elements.

Animations can be used to create engaging and interactive effects on a web page, such as transitions, fades, rotations, and more. They can also be used to add visual interest and enhance the user experience. CSS Animations are typically defined using the **@keyframes** rule, which allows you to specify different stages of animation and the CSS properties that should be applied at each stage.

Some of the benefits of using CSS Animations include:

- They are lightweight and don't require additional JavaScript or other libraries
- They can be easily customized and adjusted to fit your specific needs
- They are supported by all modern web browsers
- They can help improve the overall user experience and engagement on a website.

When using CSS Animations, it's important to consider factors such as performance, browser compatibility, and accessibility. Animations should be used thoughtfully and sparingly, and should not interfere with the functionality of the website or make it difficult for users to navigate or interact with the content.

# Applying CSS Animations

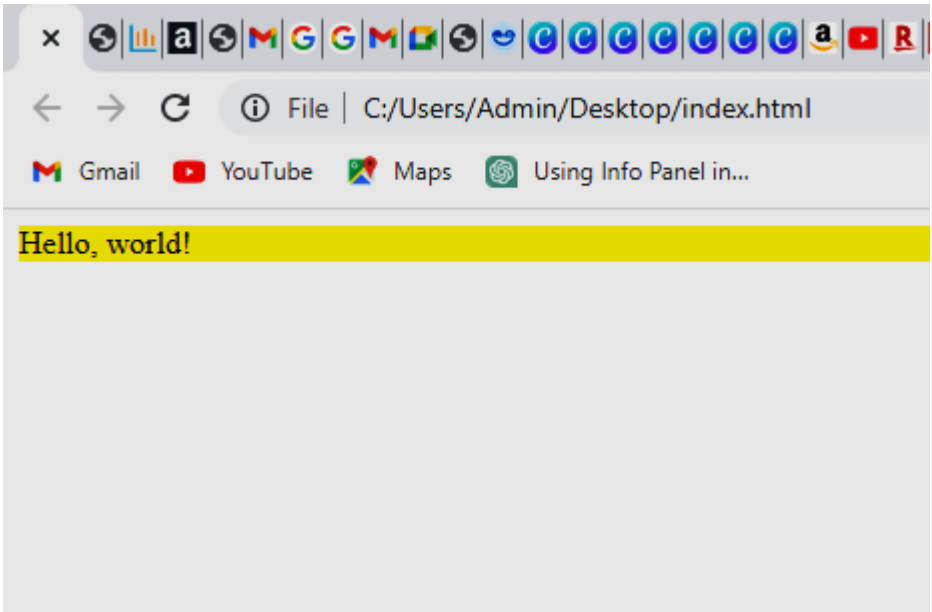
To apply CSS Animations, you need to follow these steps:

- Define the keyframes: Keyframes are used to define the animation. You can define the animation at different stages using percentage values.
- Assign the keyframes to an element: You can assign the keyframes to an element using the animation-name property.
- Set the duration and timing function: You can set the duration of the animation and timing function using the animation-duration and animation-timing-function properties.
- Specify the animation direction and iteration count: You can specify the direction of the animation and the number of times the animation should be repeated using the animation-direction and animation-iteration-count properties.

Here is an example of how to apply a simple CSS animation:

```
<!DOCTYPE html>
<html>
<head>
    <meta charset='utf-8'>
    <meta http-equiv='X-UA-Compatible' content='IE=edge'>
    <title>Example Page</title>
    <meta name='viewport' content='width=device-width, initial-scale=1'>
    <link rel='stylesheet' type='text/css' media='screen' href='main.css'>
    <script src='main.js'></script>
 <style>
      /* Define the keyframes */
      @keyframes example {
        0%   {background-color: red;}
        50%  {background-color: yellow;}
        100% {background-color: blue;}
      }

      /* Assign the keyframes to an element */
      div {
        animation-name: example;
        animation-duration: 4s;
        animation-timing-function: ease;
        animation-delay: 2s;
        animation-iteration-count: infinite;
        animation-direction: alternate;
      }
    </style>
  </head>
  <body>
    <div>Hello, world!</div>


</body>
</html>
```

In this example, we define a keyframe called **example** that changes the background color of the element from red to yellow to blue throughout the animation. We then assign this keyframe to a **div** element and specify the duration, timing function, delay, iteration count, and direction of the animation using the relevant CSS properties. See the web result below.




# CSS transitions

CSS transitions are a way to create smooth and gradual animations between two states of an element. They allow you to define how an element should change its properties over a specified duration and timing function.

To apply a CSS transition, you need to first define the starting state of the element using CSS properties. Then, you can use the **transition** property to specify the duration, timing function, and other parameters of the animation.

Here's an example of a CSS transition that changes the background color of a button when it's hovered over:

**HTML Code**:

```html
<button class="btn">Hover me!</button>
```

**CSS Code**:

```css
.btn {
  background-color: blue;
  color: white;
  padding: 10px;
  border: none;
  transition: background-color 0.3s ease;
}

.btn:hover {
  background-color: red;
}
```

In this example, we define a button with a blue background color and white text. We also set the **transition** property to **background-color 0.3s ease**, which means that the transition will last for 0.3 seconds and will use the **ease** timing function.

When the button is hovered over, we apply a new background color of red using the **:hover** pseudo-class. Because of the **transition** property, the background color change will be smooth and gradual over the specified duration and timing function.

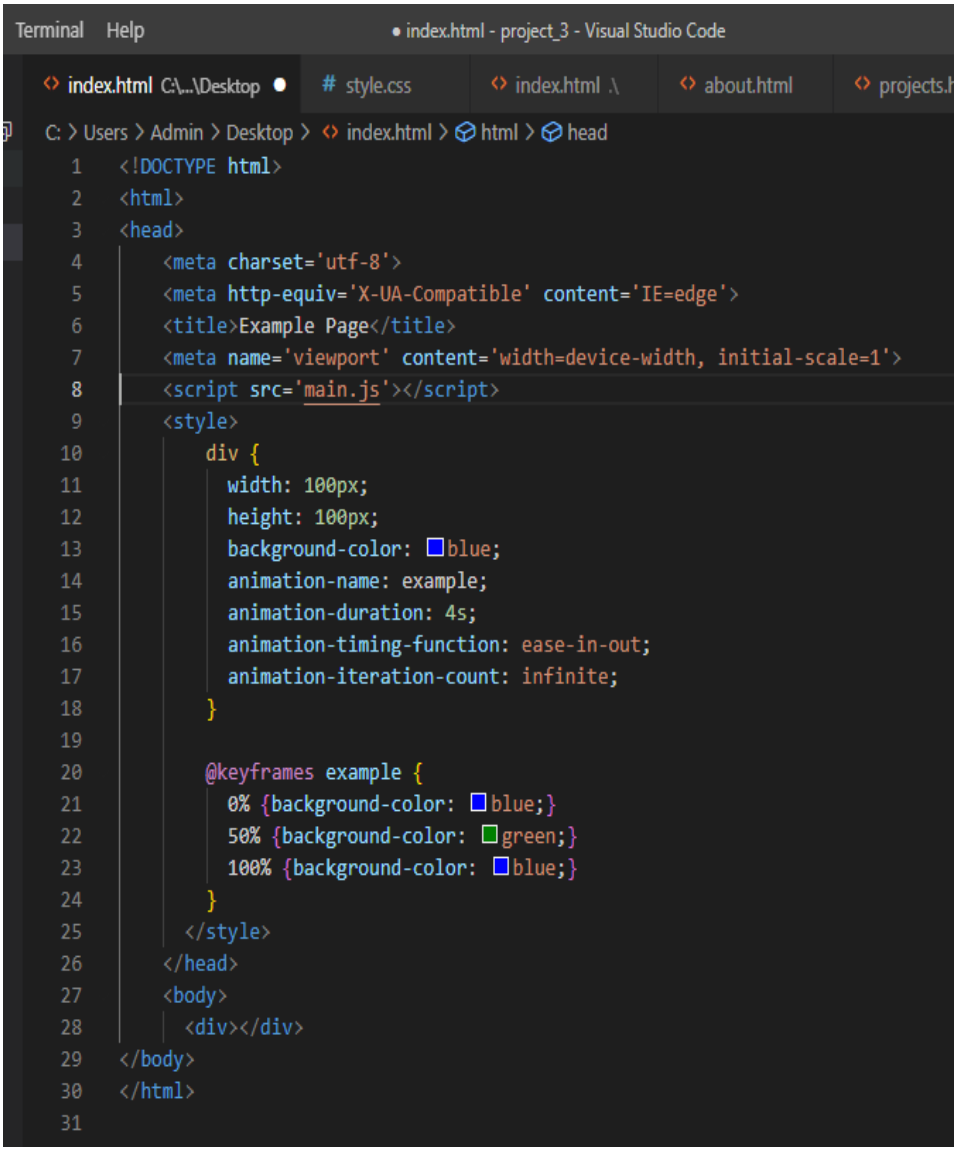
You can apply CSS transitions to a wide range of CSS properties, including background color, font size,

width, height, opacity, and more.

## Animating with Keyframes

Animating with keyframes is a CSS technique that allows you to create complex animations with more control than traditional transitions. It involves defining the animation steps at different points in time, using the **@keyframes** rule, and then applying that animation to an element using the **animation** property.

Here's an example of using keyframes to create a simple animation:




In this example, we've defined a **div** element with a blue background color and a set width and height. We then apply the **example** animation to it using the **animation-name** property. We specify the duration of the animation, the easing function, and how many times it should repeat using the **animation-duration**, **animation-timing-function**, and **animation-iteration-count** properties, respectively.

The actual animation is defined in the **@keyframes** rule. Here, we define the animation steps at different points in time: at 0%, the background color is blue, at 50% it's green, and at 100% it's blue again. By using the **animation** property to apply the **example** animation to the **div**, we get a looping animation where the background color fades between blue and green.

## Undesratanding Flexbox

Flexbox is a layout module in CSS that provides a flexible way to layout, align and distribute space among items in a container. With Flexbox, you can create flexible and responsive layouts that can adapt to different screen sizes and orientations.

Flexbox consists of two important components: the container (also known as the flex container) and the items (also known as the flex items). The container is used to group and control the layout of the items inside it. The items are the individual elements inside the container that can be arranged and aligned using flexbox properties.

Some of the key flexbox properties include:

- **display: flex** - specifies that the container should use flexbox layout
- **flex-direction** - specifies the direction of the main axis (row or column) along which the items should be laid out
- **justify-content** - specifies how the items should be aligned along the main axis (e.g. center, start, end, space-between, etc.)
- **align-items** - specifies how the items should be aligned along the cross axis (e.g. center, start, end, stretch, etc.)

- **flex-wrap** - specifies whether the items should wrap to the next line if there is not enough space on a single line
- **flex-grow** - specifies how much an item should grow relative to other items in the container
- **flex-shrink** - specifies how much an item should shrink relative to other items in the container
- **flex-basis** - specifies the initial size of an item before any remaining space is distributed.

**Applying Flexbox**

Below is an example of how to use Flexbox to create a simple navigation bar:

## HTML code:

```
<nav>
    <ul>
      <li><a href="#">Home</a></li>
      <li><a href="#">About</a></li>
      <li><a href="#">Services</a></li>
      <li><a href="#">Contact</a></li>
    </ul>
  </nav>
```

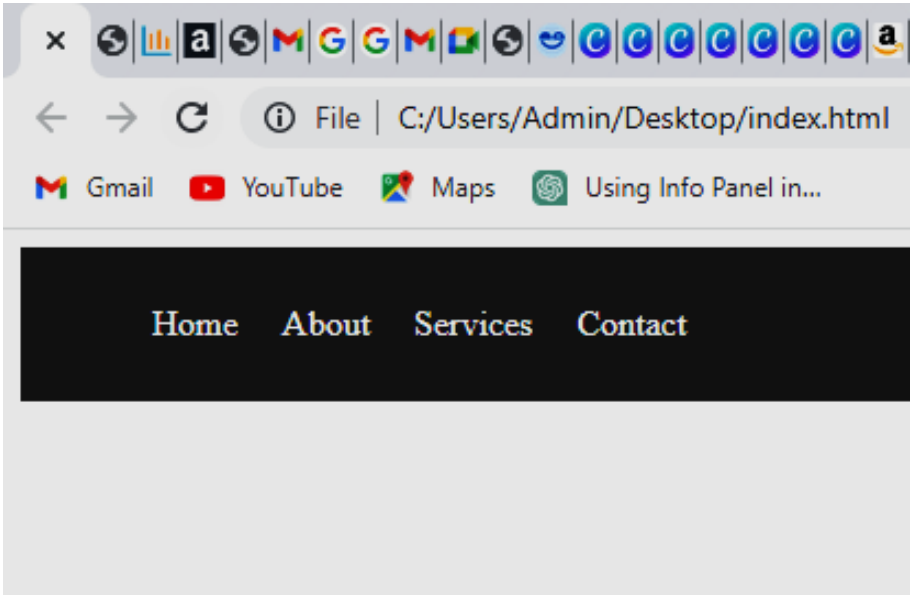
## CSS code:

```
nav {
  display: flex;
  justify-content: space-between;
  align-items: center;
  background-color: #333;
  color: #fff;
  padding: 10px;
}

ul {
  display: flex;
  list-style: none;
}
li {
  margin: 0 10px;
}
a {
  color: #fff;
  text-decoration: none;
}
```

In this example, the nav element is set to display: flex, which turns it into a flex container. The justify-content property is set to space-between, which aligns the child elements with equal space between them. The align-items property is set to center, which vertically centers the child elements. The ul element is also set to display: flex, which turns it into a flex container, and the child li elements are given a margin to separate them. Finally, the “a” element is styled with white text color and no text decoration.

This creates a horizontal navigation bar with equal spacing between the items, centered vertically within the nav element. The navigation bar is also given a

dark background color and white text color. See the result below.



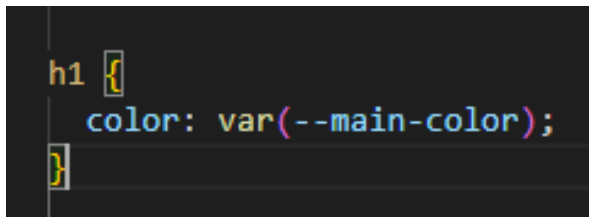

# CSS Variables

CSS variables, also known as CSS custom properties, are a powerful feature of modern CSS that allows you to define a variable once and use it throughout your CSS code. They are similar to variables in programming languages, in that they allow you to store and reuse values.

To define a CSS variable, you can use the -- syntax followed by a name and value:

```
:root {
  --main-color: #f00;
}
```

This creates a variable named --main-color with a value of #f00. You can then use this variable in other parts of your CSS code:




This sets the color of all **h1** elements to the value of the **--main-color** variable. CSS variables are useful for some reasons. For one, they allow you to define a value once and use it throughout your code, which can save you a lot of time and effort. They also allow you to easily change the value of a property across your entire website by simply updating the variable value in one place.

CSS variables can also be used in combination with JavaScript, which allows you to dynamically change the value of a variable based on user input or other events. This can be especially useful for creating dynamic and interactive websites.

# Scalable Vector Graphics (SVG)

Scalable Vector Graphics (SVG) is an XML-based vector image format for two-dimensional graphics that supports interactivity and animation. It was developed by the World Wide Web Consortium (W3C) and was first released in 2001. SVG images are resolution-

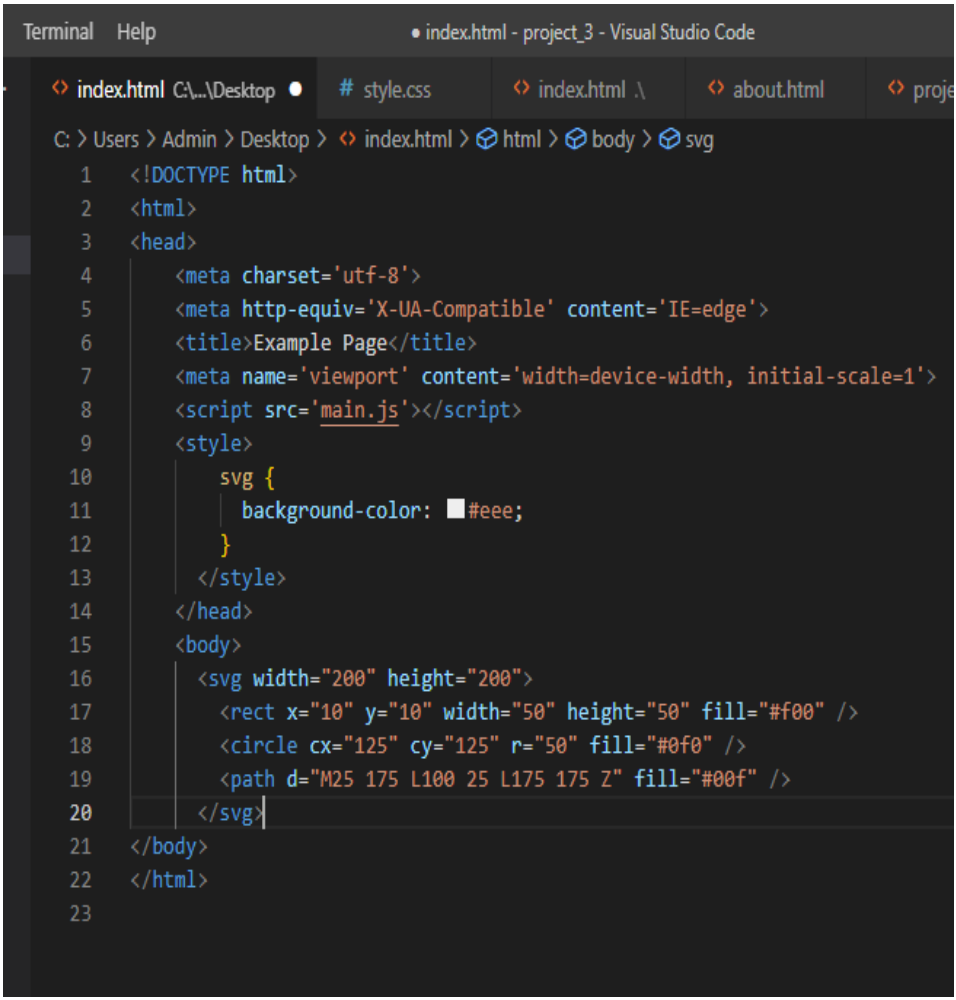
independent, meaning they can be scaled to any size without losing their quality.

SVG images are created using shapes, lines, and curves, and can be edited using text editors or specialized software such as Adobe Illustrator. They can also be created programmatically using JavaScript.

In web development, SVG is commonly used for creating logos, icons, and other graphical elements that need to be scalable and responsive. It can also be used for creating interactive graphics and animations. SVG images can be embedded directly into HTML using the <svg> element or can be included as an external file using the <img> element or CSS background property.

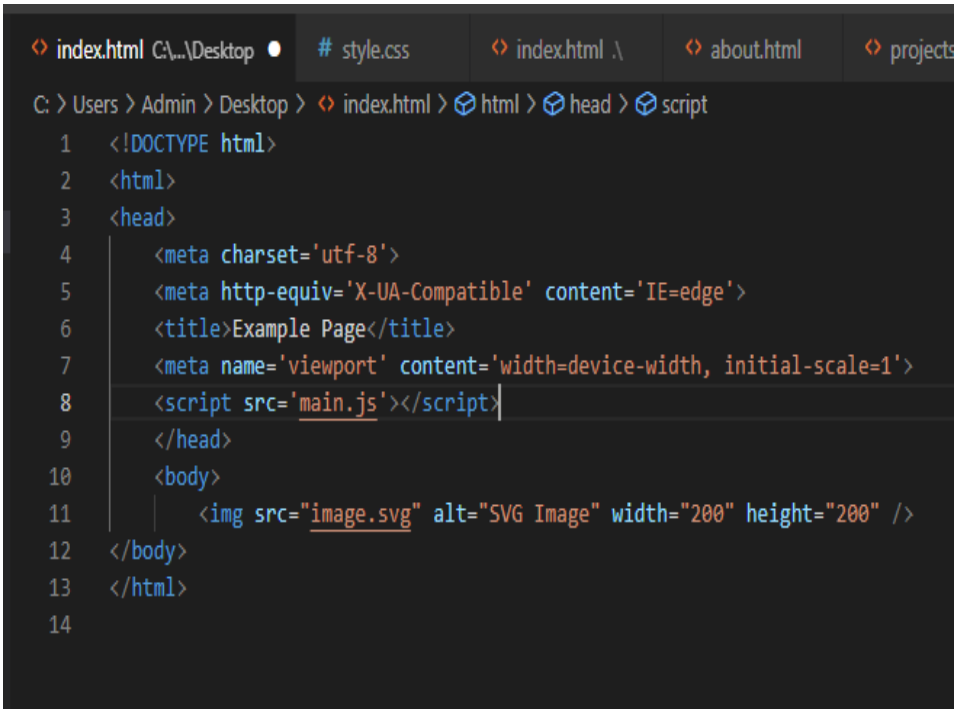
To apply Scalable Vector Graphics (SVG), you can embed SVG directly into your HTML code using the **<svg>** element, or you can include it as an external file using the **<img>** element.

To embed SVG directly, you can add the **<svg>** element to your HTML code with its required attributes, such as **width** and **height**, and then add shapes and other graphic elements inside the **<svg>** element using SVG syntax. For example:




This example creates an SVG element with a width and height of 200 pixels and a light gray background. It then adds a red square, a green circle, and a blue triangle using SVG syntax within the **<svg>** element.

To include an SVG file as an external file, you can use the **<img>** element with the **src** attribute set to the URL of the SVG file. For example:




In this example, the **<img>** element includes an SVG image with the **src** attribute set to "image.svg" and a width and height of 200 pixels.

## CSS Custom Fonts

Custom fonts in CSS refer to fonts that are not commonly available on most devices but can be imported and used on a web page. Using custom fonts can add a unique touch to the design of a website and can make it stand out from other sites that use only common fonts.

To use custom fonts in CSS, you first need to download the font files and add them to your website's directory. Then, you can use the @font-face rule to import the font into your CSS. The @font-face rule specifies the font family name and the URL of the font

file, and can also include other properties such as font-weight and font-style.

Here is an example of how to use the @font-face rule to import a custom font:

```
@font-face {
  font-family: 'MyCustomFont';
  src: url('mycustomfont.woff2') format('woff2'),
       url('mycustomfont.woff') format('woff');
  font-weight: 400;
  font-style: normal;
}

body {
  font-family: 'MyCustomFont', sans-serif;
}
```

In this example, the font family name is set to 'MyCustomFont', and the font files are located in the same directory as the CSS file. The font-weight is set to 400 (normal) and the font-style is set to normal. Finally, the font family is applied to the body element, with the fallback font being a sans-serif font.

**CSS Best Practices**

CSS Best Practices are a set of guidelines or recommendations for writing clean, maintainable, and efficient CSS code. They help developers to write better code, avoid common mistakes, and make their code more scalable and reusable.

Some common CSS best practices include:

- **Use meaningful and consistent class names**: Class names should accurately describe the content they represent and be consistent throughout the project.
- **Avoid using inline styles**: Inline styles make it difficult to maintain and update code, as they are scattered throughout the HTML markup.
- **Use a consistent indentation style**: Use consistent indentation to improve readability and make it easier to identify the structure of your code.
- **Use shorthand properties when possible**: Shorthand properties allow you to write more concise code and reduce the overall file size.
- **Keep your code organized**: Organize your code into separate files, and group related styles together.
- **Avoid using excessive specificity**: Excessive specificity makes it difficult to override styles and can lead to specificity wars.
- **Use comments to document your code**: Comments can be used to describe the purpose and function of your code and make it easier to maintain.
- **Minimize the use of !important**: The !important rule should be used sparingly, as it can make it difficult to override styles.
- **Use a CSS preprocessor**: CSS preprocessors, such as Sass or Less, can help you write more efficient and maintainable CSS code.
- **Test your code in multiple browsers**: Testing your code in multiple browsers helps to ensure cross-browser compatibility and avoid unexpected rendering issues.

# Writing Efficient CSS

Writing efficient CSS is an important skill to ensure that your web pages load quickly and smoothly. Here are some best practices for writing efficient CSS:

- **Minimize the use of selectors**: The more complex your CSS selectors are, the slower the browser will be able to render the page. Try to use the simplest selectors possible and avoid using too many descendant selectors.
- **Use shorthand properties**: Instead of using separate properties for margin, padding, and borders, use the

shorthand properties. This reduces the amount of CSS you need to write and can also make your code easier to read.

- **Avoid using too many fonts**: Using too many different fonts on a page can slow down page load time. Try to limit yourself to using two or three fonts.
- **Use CSS resets**: Browsers have their default styles for HTML elements, which can cause inconsistencies in how your web pages look. Using a CSS reset like Normalize.css can help ensure that your web pages look consistent across different browsers.
- **Group-related CSS properties**: When writing CSS, group related properties together. For example, instead of writing separate rules for margin-top, margin-bottom, margin-left, and margin-right, group them in a margin rule.
- **Use efficient CSS animations**: When using CSS animations, avoid using too many keyframes or animating too many properties at once. This can slow down page load time and make your animations less smooth.

# Debugging CSS

Debugging CSS is an important part of web development. Here are some tips for debugging CSS:

- **Use the browser's built-in developer tools**: Most modern web browsers come with built-in developer tools that allow you to inspect and debug your CSS. You can use the elements tab to view the HTML and CSS of your web page, and the console tab to view any errors or warnings.
- **Check for typos and syntax errors**: CSS is very sensitive to typos and syntax errors. Check your CSS code for any typos, missing semicolons, or other syntax errors.
- **Use meaningful class and ID names**: Use class and ID names that are descriptive and meaningful. This will make it easier to understand your CSS code and debug any issues.
- **Use comments to organize your code**: Use comments to organize your CSS code and make it easier to understand. This can also help you identify any issues or bugs in your code.
- **Use a CSS preprocessor**: CSS preprocessors like Sass or Less can help you write more efficient and organized CSS

code. They also offer features like variables and mixins that can help you avoid repeating code.

- **Use a CSS linter**: A CSS linter can help you identify and fix common CSS errors and improve the overall quality of your code.
- **Test your code in different browsers**: CSS can behave differently in different browsers, so it's important to test your code in multiple browsers to ensure it works as expected.

# The Future of CSS

The future of CSS is an exciting prospect, as new features and capabilities are constantly being developed and implemented. Here are some of the trends and developments that are shaping the future of CSS:

- **CSS Grid:** CSS Grid is a powerful layout system that allows for complex layouts to be created quickly and easily. It has already gained widespread adoption and is becoming the standard for layout design.
- **CSS Custom Properties**: CSS custom properties, or variables, allow developers to define reusable values and apply them to various parts of their CSS, making it easier to maintain and update styles.
- **CSS Animations and Transitions**: CSS animations and transitions are becoming more sophisticated, with new properties and features being added that allow for more complex and interactive animations.
- **Responsive Web Design**: Responsive design has already become the standard for web design, but as more devices and screen sizes emerge, the need for responsive design will only continue to grow.
- **Accessibility**: Accessibility is becoming increasingly important, and CSS plays a vital role in making websites accessible to all users. New CSS features and techniques are being developed to make it easier to create accessible designs.

# HTML AND CSS PRACTICAL EXERCISES

This chapter consists of HTML and CSS Practical Exercises that are intended to boost beginners and advanced web developer 'ability in building professional websites with HTML and CSS.

## Exercise 1: Design a simple login page with HTML and CSS

Below is an example of HTML and CSS code for a simple Login page.

**HTML Code:**

```
<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

        <meta http-equiv="X-UA-Compatible" content="IE=edge">

        <meta name="viewport" content="width=device-
```

```
width, initial-scale=1.0">

    <title>Login Page</title>

        <link rel="stylesheet" type="text/css"
href="style.css">

</head>

<body>

  <div class="login-container">

    <h1>Login</h1>

    <form>

      <label for="username">Username</label>

                <input type="text" id="username"
name="username">

      <label for="password">Password</label>

              <input type="password" id="password"
name="password">

       <button type="submit">Sign In</button>

    </form>

  </div>

</body>
```

```
</html>
```

## CSS Code:

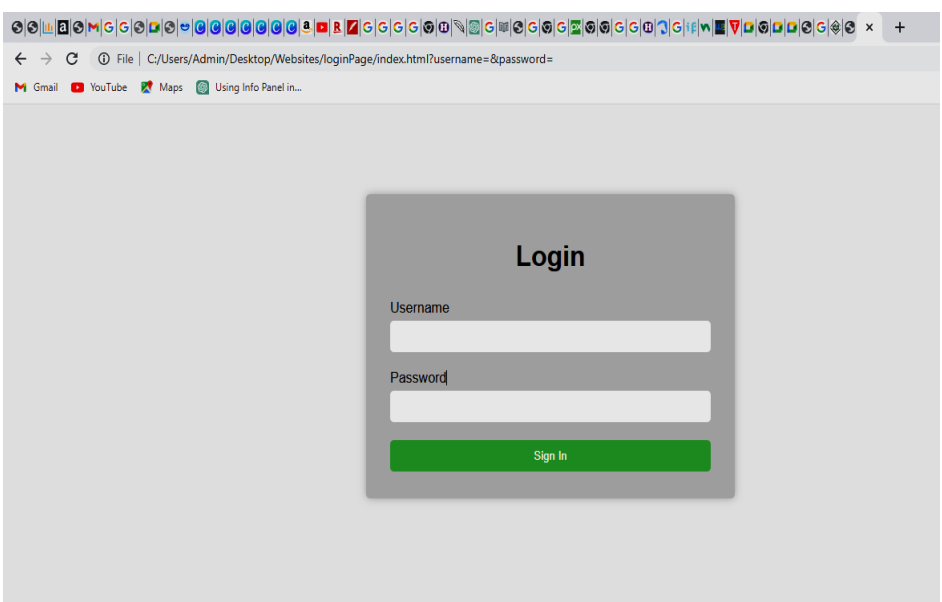
Create a CSS file and include the following code.

```
body {
  margin: 0;
  padding: 0;
  font-family: Arial, sans-serif;
  background-color: #f2f2f2;
}
.login-container {
  width: 400px;
  margin: 100px auto;
  background-color: #bbbbbb;
  padding: 30px;
  border-radius: 5px;
  box-shadow: 0px 0px 10px 0px rgba(0,0,0,0.3);
}
h1 {
```

```
  text-align: center;
  margin-bottom: 30px;
}
form {
  display: flex;
  flex-direction: column;
}
label {
  margin-bottom: 5px;
}
input {
  padding: 10px;
  margin-bottom: 20px;
  border: none;
  border-radius: 5px;
}
button {
  background-color: #4CAF50;
```

```
  color: white;
  padding: 10px;
  border: none;
  border-radius: 5px;
  cursor: pointer;
}
button:hover {
  background-color: #3e8e41;
}
```

The web result is shown below.



# Exercise 2: Build a simple landing page for a product with HTML and CSS.

Below is an example of HTML and CSS code for a simple landing page for a product:

**HTML code:**

```
<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

        <meta http-equiv="X-UA-Compatible" content="IE=edge">

    <meta name="viewport" content="width=device-width, initial-scale=1.0">

    <title>Product Landing Page</title>

  <link rel="stylesheet" href="style.css">

</head>

<body>

  <header>
```

```
    <nav>
      <ul>
        <li><a href="#">Home</a></li>
        <li><a href="#">Features</a></li>
        <li><a href="#">Pricing</a></li>
        <li><a href="#">Contact</a></li>
      </ul>
    </nav>
    <h1>Mouse</h1>
  </header>
  <main>
    <section>
      <h2>Features</h2>
      <ul>
        <li>Rechageable</li>
        <li>8hrs Battery Life</li>
        <li>Grey Color</li>
      </ul>
```

```
    </section>
    <section>
     <h2>Pricing</h2>
     <p>Price: $20</p>
      <button>Buy Now</button>
    </section>
  </main>
  <footer>
   <p>Copyright; 2023 Sammie Smith. All rights reserved.</p>
  </footer>
</body>
</html>
```

**CSS code:**

Create a CSS file and include the following code.

```
body {
    font-family: Arial, sans-serif;
    margin: 0;
    padding: 0;
```

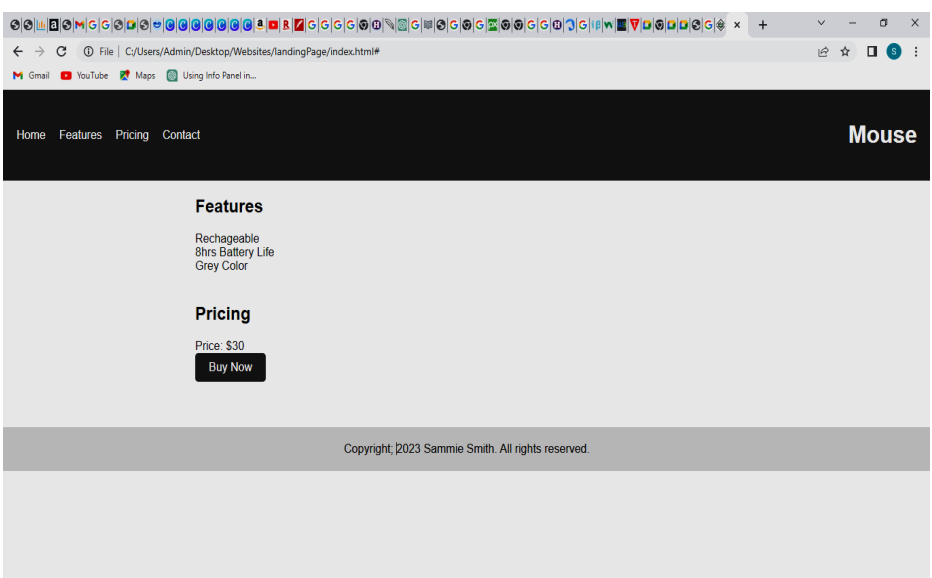
```
}

  header {

  background-color: #333;

  color: white;

  padding: 20px;

  display: flex;

  justify-content: space-between;

 align-items: center;

}

  nav ul {

  list-style: none;

  margin: 0;

  padding: 0;

  display: flex;

}

  nav li {

  margin-right: 20px;

}
```

```
  nav li:last-child {
  margin-right: 0;
}
  nav a {
  color: white;
 text-decoration: none;
}
 main {
  padding: 20px;
  max-width: 800px;
  margin: 0 auto;
}
  section {
  margin-bottom: 40px;
}
 h2 {
  margin-top: 0;
}
```

```
ul {
  list-style: none;
  margin: 0;
  padding: 0;
}
p {
  margin: 0;
}
 button {
  background-color: #333;
  color: white;
  border: none;
  padding: 10px 20px;
  border-radius: 5px;
  cursor: pointer;
  font-size: 16px;
}
 button:hover {
```

```
  background-color: #666;
}
footer {
  background-color: #ccc;
  padding: 20px;
  text-align: center;
}
```

This code creates a simple landing page with a navigation bar, product name, features section, pricing section, and footer. You can customize the content and styling to fit your product's needs. See the web result below.




Note that you can modify the exercise above to suite your project.

# Exercise 3: Create a responsive pricing table with HTML and CSS

Below is an example of a responsive pricing table using HTML and CSS:

**HTML Code:**

```
<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

        <meta http-equiv="X-UA-Compatible" content="IE=edge">

    <meta name="viewport" content="width=device-width, initial-scale=1.0">

    <title>Pricing Table</title>

        <link rel="stylesheet" type="text/css" href="style.css">

  </head>

  <body>

    <div class="pricing-table">
```

```
<div class="plan">
<h2>Basic</h2>
<h3>$10/month</h3>
<ul>
  <li>1 User</li>
  <li>5GB Storage</li>
  <li>Unlimited Projects</li>
  <li>Free Support</li>
</ul>
  <a href="#" class="button">Choose Plan</a>
</div>

<div class="plan popular">
<h2>Pro</h2>
<h3>$20/month</h3>
<ul>
  <li>5 Users</li>
  <li>20GB Storage</li>
```

```
      <li>Unlimited Projects</li>
      <li>Free Support</li>
    </ul>
     <a href="#" class="button">Choose Plan</a>
    </div>
    <div class="plan">
    <h2>Advanced</h2>
    <h3>$30/month</h3>
    <ul>
      <li>Unlimited Users</li>
      <li>50GB Storage</li>
      <li>Unlimited Projects</li>
      <li>Free Support</li>
    </ul>
     <a href="#" class="button">Choose Plan</a>
    </div>
   </div>
</body>
```

```
</html>
```

## CSS Code:

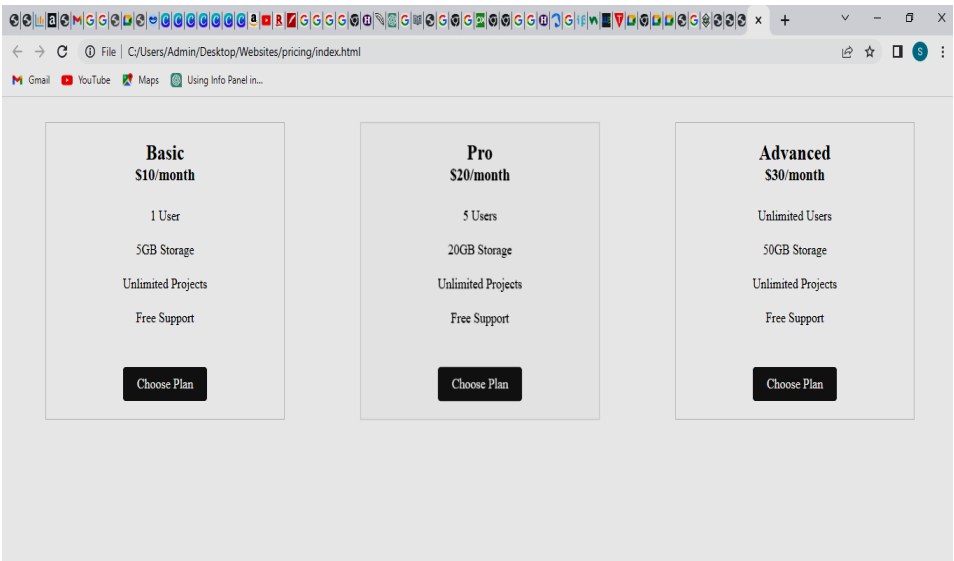
Create a CSS file and include the following code.

```
.pricing-table {
    display: flex;
    justify-content: space-around;
    flex-wrap: wrap;
}
.plan {
    text-align: center;
    margin: 20px;
    padding: 20px;
    border: 1px solid #ccc;
    width: 30%;
    max-width: 300px;
    transition: all 0.3s ease-in-out;
}
.plan:hover {
```

```
  transform: scale(1.1);
  box-shadow: 0 5px 10px rgba(0,0,0,0.3);
}
.popular {
  background-color: #f9f9f9;
}
h2, h3 {
  margin: 0;
}
ul {
  list-style: none;
  padding: 0;
}
li {
  padding: 10px 0;
}
.button {
  display: inline-block;
```

```
    padding: 10px 20px;
    margin-top: 20px;
    background-color: #333;
    color: #fff;
    text-decoration: none;
    border-radius: 4px;
}
```

The web result is shown as follows.



**CHAPTER TEN**

**RESPONSIVE WEB DESIGN**

Responsive web design is an approach to web design that focuses on creating websites that are optimized for a variety of devices and screen sizes. With the increasing popularity of mobile devices, responsive web design has become essential for ensuring that websites are accessible and user-friendly on all devices.

Responsive web design involves using CSS media queries to adjust the layout and content of a website based on the size of the screen it is being viewed on. This allows for a seamless user experience, regardless of the device being used to access the website.

In this chapter, we will explore the principles of responsive web design, including best practices for designing responsive websites and the tools and technologies used to create them. We will also cover some common challenges and solutions associated with responsive web design, such as optimizing images and managing complex layouts.

# Introduction to Responsive Design

Responsive design is an approach to web design that ensures websites and applications look good and function well across multiple devices and screen sizes.

With the growing use of mobile devices to access the internet, responsive design has become a critical aspect of modern web development.

Responsive design typically involves using a combination of techniques, such as fluid layouts, flexible images, and media queries, to create websites that can adapt to different screen sizes and resolutions. The goal of responsive design is to provide a seamless and consistent user experience, regardless of the device being used to access the website. This means that elements on the page will reposition, resize, or even disappear based on the screen size and orientation, without losing any functionality or readability.

Responsive design ensures that websites and applications are accessible to all users, regardless of the device they are using and helps to ensure that they can navigate and interact with the content in the most efficient and user-friendly way possible.

# Best Practices for Designing Responsive Websites

Below are some best practices for designing responsive websites:

- **Mobile-first approach**: Start by designing for the smallest screen size first and then gradually move towards larger screen sizes. This ensures that the website is optimized for smaller screens and loads faster.
- **Use a responsive framework**: Responsive frameworks like Bootstrap provide a grid system that helps in creating

responsive layouts quickly. This also ensures that the website is compatible with different screen sizes.

- **Optimize images**: Large images can slow down the website's loading time, so it's important to optimize them for web use. You can use image compression tools like TinyPNG or JPEGmini to reduce the size of images without affecting their quality.
- **Use media queries:** Media queries allow you to apply different styles to different screen sizes. This helps in creating a consistent user experience across different devices.
- **Test on multiple devices**: It's important to test the website on different devices to ensure that it looks and functions correctly on all of them. You can use tools like BrowserStack or Device Mode in Google Chrome to test the website on different devices.
- **Keep it simple**: Avoid using too many complex design elements or animations that can slow down the website's loading time. Stick to simple and clean design principles that are easy to read and navigate on smaller screens.
- **Use scalable fonts**: Use fonts that are scalable and easy to read on smaller screens. Avoid using fonts that are too small or difficult to read on smaller screens.

# Tools and Technologies used to Create Responsive Website

There are several tools and technologies used to create responsive websites. Some of the most popular ones include:

- **HTML and CSS**: These are the basic building blocks of any website and are used to create the structure and style of the website.
- **CSS frameworks**: Frameworks like Bootstrap and Foundation provide pre-built CSS styles and components that make it easier to create responsive websites.
- **CSS preprocessors**: Preprocessors like Sass and Less

allow developers to write CSS more efficiently by providing features like variables, nesting, and mixins.

- **JavaScript**: JavaScript can be used to add interactivity and dynamic behavior to a website. It is often used for things like responsive navigation menus and sliders.
- **Responsive design tools**: There are many tools available that help with responsive design, such as media query generators, viewport testing tools, and responsive design frameworks.
- **Content management systems (CMS):** CMS platforms like WordPress, Drupal, and Joomla have built-in responsive design features and templates that can be used to create responsive websites.
- **Mobile app development platforms**: Platforms like React Native and Flutter allow developers to create mobile apps that are optimized for different devices and screen sizes.

## Understanding Media Queries

Media queries are a CSS technique used for styling web pages based on the device or screen size they are viewed on. With media queries, you can define different styles for different devices or screen sizes, such as desktops, laptops, tablets, and mobile devices.

Media queries work by checking the width and height of the viewport, which is the area of the browser window where the web page is displayed. You can set a specific style for a particular viewport size or range of sizes by using the **@media** rule in CSS.

For example, you could define a media query for screens with a maximum width of 768 pixels and set a different font size and layout for that size range. This allows you to create a more user-friendly and consistent experience across different devices.

Media queries can also be used to apply different styles based on other characteristics of the device, such as orientation (landscape or portrait), resolution, or color depth.

**Applying Media Queries**

To apply media queries in CSS, you first need to specify the media type you want to apply the styles to. The most common media type is the screen, which is used for devices with a screen, such as desktop computers, laptops, tablets, and smartphones. Other media types include print, which is used for printing, and speech, which is used for screen readers.

To apply styles to a specific media type, you can use the @media rule followed by the media type in parentheses, like this:

Within the media query block, you can write styles that will only apply when the specified media type is in use. For example, you can set different font sizes, colors, or layout properties for desktop and mobile devices.

You can also use media queries to specify different styles based on the width or height of the device's screen. For example, you can write styles that will only

apply when the screen width is less than or equal to 768 pixels:

```
@media screen and (max-width: 768px) {
  /* styles go here */
}
```

This will apply the styles within the media query block when the screen width is 768 pixels or less. You can also use the **min-width** property to specify a minimum screen width for the styles to apply.

By using media queries, you can create responsive designs that adjust to different screen sizes and devices, making your website more user-friendly and accessible.

### Understanding Mobile-first Design Approach

Mobile-first design is a design approach in which a website or application is designed for mobile devices first and then scaled up to fit larger screens such as desktops or laptops. This approach assumes that more people access websites through their mobile devices than their desktops, and it aims to create an optimal experience for mobile users.

When designing for mobile first, the focus is on creating a streamlined, efficient user experience that prioritizes the most important content and tasks. This often involves using a minimalist design, simplifying navigation, and using large, easy-to-read fonts and

buttons.

Once the mobile design is in place, the design can be scaled up to fit larger screens. This involves using media queries to adjust the layout and styling of the website based on the screen size of the device being used.

Mobile-first design has become increasingly important as mobile devices have become the primary way that people access the internet. By designing with mobile in mind, developers can ensure that their websites are accessible and usable for the majority of their audience.

**Applying Mobile-first Design**

To carry out a mobile-first design using CSS, follow these steps:

- Start with a mobile-first CSS file, which contains styles for the smallest screen sizes first, and then gradually adds styles for larger screen sizes as necessary. For example:

```css
/* Styles for mobile devices */
@media (min-width: 320px) {
  /* Styles for screens larger than 320px */
}

@media (min-width: 480px) {
  /* Styles for screens larger than 480px */
}

@media (min-width: 768px) {
  /* Styles for screens larger than 768px */
}

@media (min-width: 1024px) {
  /* Styles for screens larger than 1024px */
}
```

- Use responsive units such as **em** and **%** instead of fixed units like **px** to ensure that elements scale properly on different screen sizes.
- Use a responsive framework such as Bootstrap or Foundation, which provide pre-built CSS classes and components for creating responsive layouts.
- Use CSS media queries to target specific screen sizes and apply custom styles as needed.
- Test your design on various devices and screen sizes to ensure that it looks good and functions properly across all platforms.

Study the code below to understand how to implement Mobile-first Design.

**HTML Code:**

```
<!DOCTYPE html>

<html>

<head>

    <meta charset='utf-8'>

                <meta http-equiv='X-UA-Compatible' content='IE=edge'>

    <title>Example Page</title>

        <meta name='viewport' content='width=device-width, initial-scale=1'>

     <link rel='stylesheet' type='text/css' media='screen' href='main.css'>

    <script src='main.js'></script>

</head>

<body>

<header>

            <h1>My Website</h1>

             <nav>

                <ul>

                    <li><a href="#">Home</a></li>

                    <li><a href="#">About</a></li>
```

```
            <li><a href="#">Contact</a></li>
        </ul>
    </nav>
</header>
<main>
    <section>
        <h2>About Me</h2>
        <p>Hi, my name is John Doe and I am a web developer.</p>
    </section>
    <section>
        <h2>Recent Work</h2>
        <div class="work">
            <a href="#"><img src="work1.jpg" alt="Work 1"></a>
            <a href="#"><img src="work2.jpg" alt="Work 2"></a>
            <a href="#"><img src="work3.jpg" alt="Work 3"></a>
        </div>
```

```
        </section>

    </main>

    <footer>

        <p> Copyrights @ 2023 My Website. All Rights Reserved.</
p>

    </footer>

</body>

</html>
```

Afterward, create a CSS external file and link it to your html file and include the code below. Ensure to use information suitable to the project you are working on.

**CSS Code:**

```
* {

    box-sizing: border-box;

}

body {

    font-family: Arial, sans-serif;

    font-size: 16px;

    margin: 0;

    padding: 0;
```

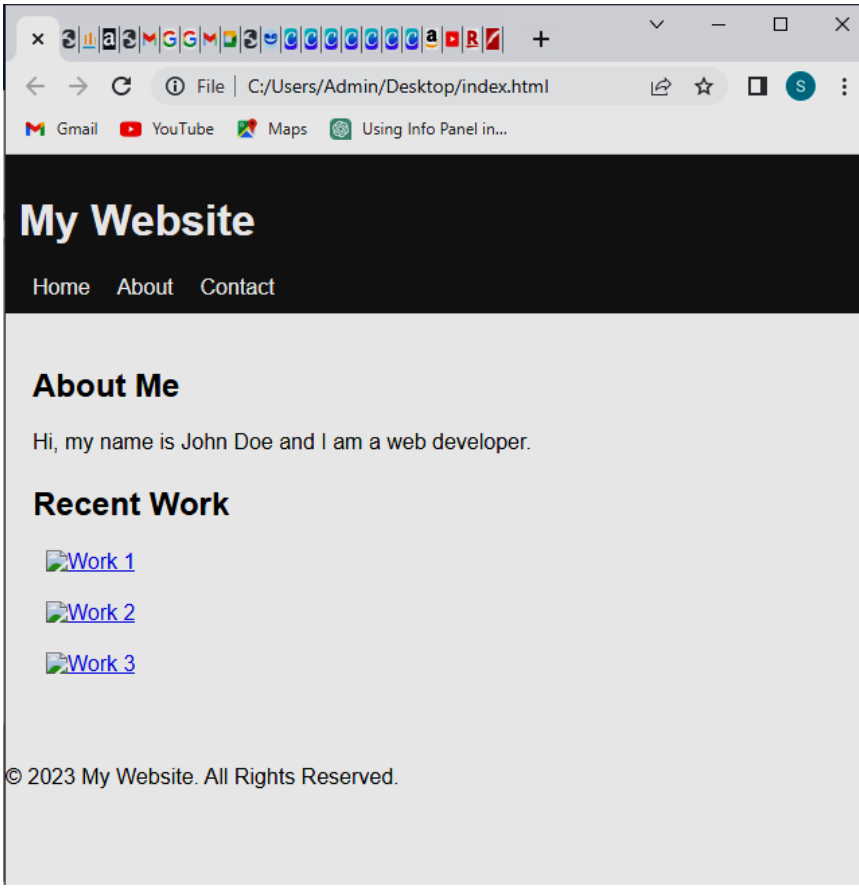
```
}
header {
    background-color: #333;
    color: #fff;
    padding: 10px;
}
nav ul {
    list-style: none;
    margin: 0;
    padding: 0;
    display: flex;
    flex-wrap: wrap;
}
nav li {
    margin: 0 10px;
}
nav a {
    color: #fff;
```

```
    text-decoration: none;
}
main {
    padding: 20px;
}
section {
    margin-bottom: 20px;
}
section h2 {
    font-size: 24px;
    margin-bottom: 10px;
}
section p {
    font-size: 16px;
    line-height: 1.5;
}
.work {
    display: flex;
```

```
        flex-wrap: wrap;
}
.work a {
        flex: 1 1 300px;
        margin: 10px;
}
@media screen and (min-width: 768px) {
        header {
            display: flex;
            justify-content: space-between;
            align-items: center;
        }
        nav {
            flex: 1;
        }
        main {
            display: flex;
            flex-wrap: wrap;
```

```
    }

    section {

        flex: 1;

        margin: 0 10px;

    }

    .work a {

        margin: 10px 0;

    }

}
```

This code includes a simple header, navigation, main content with two sections, and a footer. The CSS includes styles for mobile devices with a small screen width of fewer than 768 pixels, as well as larger screen sizes with a media query. When the screen width is at least 768 pixels, the header, and main content are displayed using a flexbox layout. See the result below.




Note that you can always modify the codes in this guide to suit your taste.

# Responsive Frameworks

Responsive frameworks are pre-built collections of CSS and JavaScript files that provide a foundation for creating responsive web designs. They typically include a grid system and pre-defined styles for common HTML elements, as well as pre-built components like navigation menus, buttons, and forms.

Using a responsive framework can greatly simplify the process of building responsive websites, as it eliminates the need to write custom CSS for every design element. Instead, developers can use the framework's pre-built classes and components to quickly create responsive layouts and designs.

Some popular responsive frameworks include Bootstrap, Foundation, and Bulma. These frameworks are widely used and well-documented, making them a great choice for both beginner and experienced developers.

Note that to make a completely interactive website with a powerful feature you would require all the frameworks put together, but this book is limited to HTML and CSS.

# Common Challenges and Solutions Associated with Responsive Web Design

Responsive web design can present some challenges that need to be addressed to provide a seamless experience across all devices. Some common challenges and solutions associated with responsive web design include:

- Device compatibility: With the wide variety of devices available today, ensuring that your website looks good on all of them can be a challenge. The solution is to use responsive design techniques that adapt to different screen sizes, such as flexible grids, responsive images, and media

queries.

- Content prioritization: When designing for smaller screens, it's important to prioritize content so that the most important information is displayed first. One solution is to use a mobile-first approach, where you design for the smallest screen size first and then add more content and features as the screen size increases.
- Navigation: Navigation menus can be tricky on small screens, as they can take up valuable screen real estate. A solution is to use a hamburger menu, where the menu is hidden behind a small icon until the user taps on it.
- Performance: Large images and complex layouts can slow down the performance of your website, especially on mobile devices with slower internet connections. A solution is to optimize images and use techniques such as lazy loading to improve page speed.
- Testing: Testing your website on different devices and screen sizes can be time-consuming and difficult. A solution is to use responsive design testing tools that allow you to preview your website on different devices and screen sizes, such as BrowserStack, Responsinator, or Am I Responsive.

## Optimizing Images and Managing Complex Layouts

As part of the process of building responsive websites, optimizing images and managing complex layouts are two critical areas to focus on.

Optimizing images involves reducing the file size of images to improve website performance and load times. This can be achieved through techniques such as compressing images, resizing images to the appropriate dimensions, and using the appropriate image file format. There are also various tools available to help optimize images, such as Photoshop and online tools like TinyPNG.

Managing complex layouts involves creating layouts

that work across a variety of screen sizes and devices. This can be achieved through the use of CSS grid systems and frameworks, which provide pre-defined grids and responsive design patterns that can be easily adapted to different screen sizes. Additionally, designing with a mobile-first approach can help ensure that the most important content is prioritized for smaller screens, with additional content and layout elements added as the screen size increases.

Other common challenges in responsive web design include issues with typography, navigation, and performance optimization. However, with careful planning and attention to detail, these challenges can be overcome, resulting in a well-designed and functional responsive website.

## CHAPTER ELEVEN

## RESPONSIVE WEBSITES PRACTICAL EXERCISES

In this chapter, we will focus on practical exercises that will help you create a responsive website from scratch. You will learn how to use HTML and CSS to build a website that looks good on desktop computers, tablets, and smartphones.

We will cover various topics such as layout design, media queries, and responsive images. By the end of this chapter, you will be able to create a website that looks great on any device, making it accessible to a wider audience.

# Exercise 1: Design a responsive restaurant website

Design a responsive restaurant website with a homepage, menu page, reservation page, and contact page. Use responsive design techniques to ensure the website looks good on all devices and provides a user-friendly experience for customers with HTML and CSS.

Below is an example of how you can create a restaurant website.

**HTML Code:**

```
<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

                <meta        http-equiv="X-UA-Compatible"
content="IE=edge">

       <meta   name="viewport"   content="width=device-
width, initial-scale=1.0">

    <title>Restaurant Homepage</title>

  <link rel="stylesheet" href="style.css">

</head>

<body>



  <header>

    <nav>

     <ul>

       <li><a href="#">Home</a></li>

       <li><a href="#">Menu</a></li>

       <li><a href="#">Reservation</a></li>
```

```
    <li><a href="#">Contact</a></li>
  </ul>
 </nav>
</header>
<main>
 <section class="hero">
  <h1>Welcome to Our Restaurant</h1>
  <p>Enjoy delicious food and great ambiance</p>
  <a href="#">See Menu</a>
 </section>
 <section class="menu">
  <h2>Our Menu</h2>
  <ul>
   <li>
      <img src="img1.jpeg" alt="Pizza">
    <h3>Pizza</h3><br>
    <p>Delicious pizza with fresh toppings</p>
    <span>$10</span>
```

```
    </li>
    <li>
        <img src="img2.jpeg" alt="Burger">
        <h3>Burger</h3>
        <p>Classic burger with all the fixings</p>
        <span>$8</span>
    </li>
    <li>
        <img src="img3.jpeg" alt="Pasta">
        <h3>Pasta</h3>
        <p>Freshly made pasta with delicious sauce</p>
        <span>$12</span>
    </li>
</ul>
</section>
<section class="reservation">
<h2>Make a Reservation</h2>
<form>
```

```
    <label for="name">Name:</label>
    <input type="text" id="name" name="name">
    <label for="email">Email:</label>
    <input type="email" id="email" name="email">
    <label for="date">Date:</label>
    <input type="date" id="date" name="date">
    <label for="time">Time:</label>
    <input type="time" id="time" name="time">
    <label for="guests">Number of Guests:</label>
    <input type="number" id="guests" name="guests">
    <button type="submit">Submit</button>
  </form>
</section>
<section class="contact">
  <h2>Contact Us</h2>
  <form>
    <label for="name">Name:</label>
```

```
        <input type="text" id="name" name="name">
        <label for="email">Email:</label>
        <input type="email" id="email" name="email">
        <label for="message">Message:</label>
        <textarea id="message" name="message"></textarea>
        <button type="submit">Submit</button>
    </form>
    </section>
</main>
<footer>
    <p>&copy; 2023 Yummies Restaurant</p>
</footer>
</body>
</html>
```

## CSS Code:

```
/* Global styles */
* {
    box-sizing: border-box;
```

```
  margin: 0;
  padding: 0;
}
  body {
  font-family: Arial, sans-serif;
}
  header {
  background-color: #333;
  color: #fff;
  padding: 10px;
}
  nav ul {
  list-style: none;
  margin: 0;
  padding: 0;
}
  nav li {
  display: inline-block;
```

```
  margin-right: 20px;
}
  nav a {
  color: #fff;
  text-decoration: none;
}
  main {
  padding: 20px;
}
  section {
  margin-bottom: 40px;
}
  h1 {
  font-size: 48px;
  margin-bottom: 20px;
}
  h2 {
  font-size: 36px;
```

```
  margin-bottom: 10px;
}
  ul {
   list-style: none;
   margin: 0;
   padding: 0;
}
  li {
   display: flex;
   margin-bottom: 20px;
}
  img {
   margin-right: 20px;
}
  span {
   margin-left: auto;
  font-weight: bold;
}
```

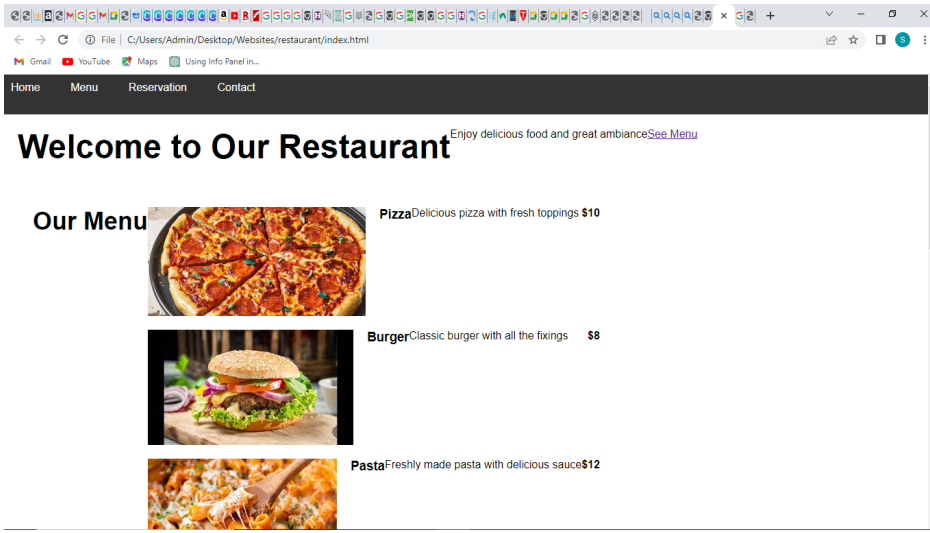
```
 form {
  display: flex;
  flex-direction: column;
}
 label {
  margin-bottom: 10px;
}

input,
textarea,
button {
  padding: 10px;
  margin-bottom: 20px;
  border: 1px solid #ccc;
  border-radius: 5px;
}
 input:focus,
textarea:focus {
```

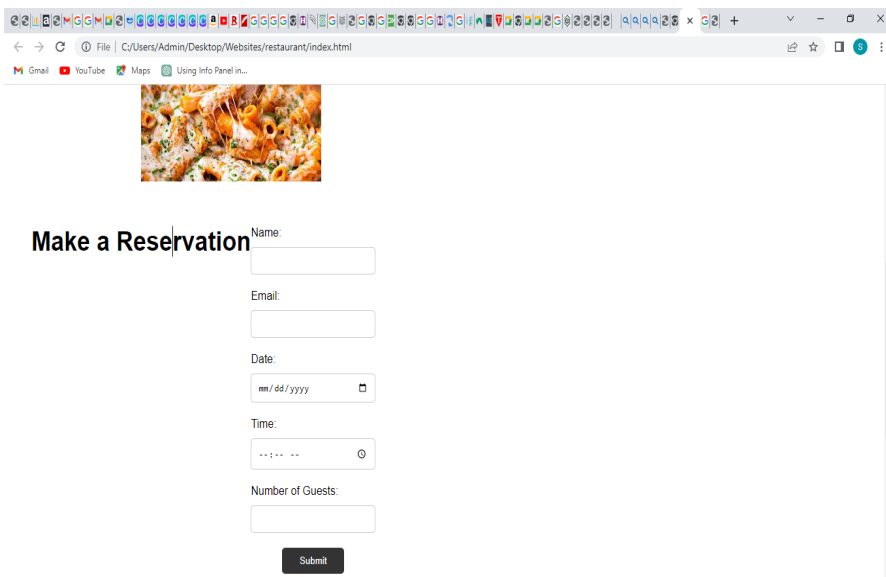
```
  outline: none;
  border-color: #333;
}
  button {
  background-color: #333;
  color: #fff;
  border: none;
  cursor: pointer;
}
  button:hover {
  background-color: #555;
}
  footer {
  background-color: #333;
  color: #fff;
  text-align: center;
  padding: 10px;
}
```

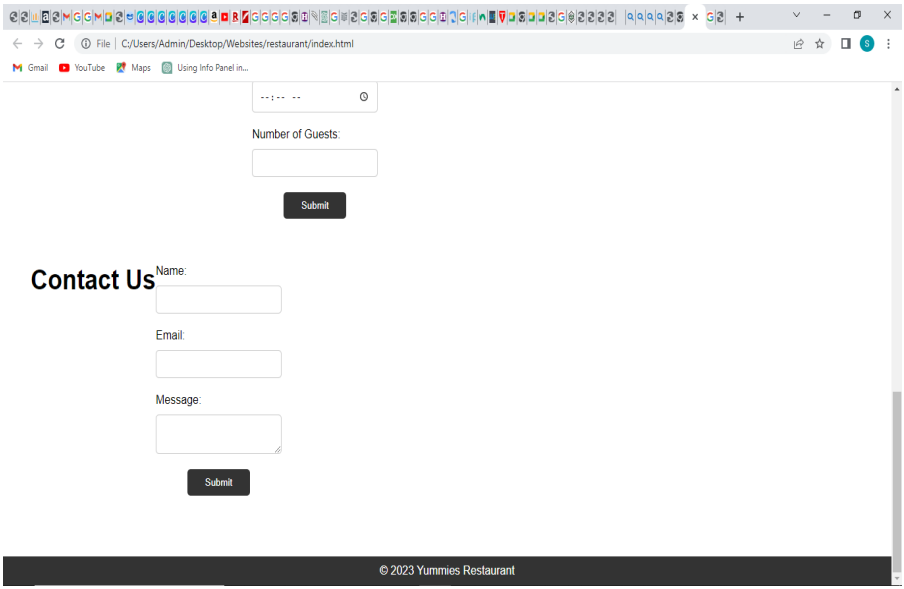
```
 /* Media queries */
 @media (min-width: 768px) {
/* Header */
 nav li {
   margin-right: 40px;
}
  /* Sections */
 section {
   display: flex;
  flex-wrap: wrap;
}
  .hero {
  flex-basis: 50%;
   padding-right: 40px;
}
  .menu,
.reservation,
.contact {
```

```
    flex-basis: 27%;
    padding-left: 22px;
    padding-right: 22px;
  }
  /* Form fields */
  label,
  input,
  textarea,
  button {
    width: 100%;
  }
  /* Form buttons */
  button {
    width: 50%;
    margin-left: auto;
    margin-right: auto;
  }
}
```

See the web results below.






# Exercise 2: Create a Responsive Photographer Portfolio

Create a responsive online portfolio for a photographer, artist, or designer. Showcase their work in a visually appealing way, using responsive design techniques to ensure the website looks good on all devices with HTML and CSS.

Below is an example of HTML and CSS code to create a responsive online portfolio for a photographer:

## HTML Code:

*<!DOCTYPE html>*

*<html lang="en">*

```
<head>
  <meta charset="UTF-8">
  <meta http-equiv="X-UA-Compatible" content="IE=edge">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Photographer Portfolio</title>
  <link rel="stylesheet" href="style.css">
</head>
<body>
  <header>
    <nav>
      <ul>
        <li><a href="#">Home</a></li>
        <li><a href="#">Portfolio</a></li>
        <li><a href="#">About</a></li>
        <li><a href="#">Contact</a></li>
      </ul>
    </nav>
```

```
</header>
<main>
  <section class="hero">
  <h1>Welcome to My Portfolio</h1>
  <p>Check out my latest work</p>
  </section>
  <section class="portfolio">
  <h2>My Portfolio</h2>
  <ul>
    <li>
        <img src="img7.jpg" alt="Photograph 1">
      <h3>Photograph 1</h3>
    </li>
    <li>
        <img src="img5.jpeg" alt="Photograph 2">
      <h3>Photograph 2</h3>
    </li>
    <li>
```

<img src="img6.jpeg" alt="Photograph 3">

<h3>Photograph 3</h3>

</li>

<li>

<img src="img4.jpeg" alt="Photograph 4">

<h3>Photograph 4</h3>

</li>

</ul>

</section>

<section class="about">

<h2>About Me</h2>

Hi, I'm a photographer based in New York City. I specialize in landscape and portrait photography. My goal is to capture the beauty of the world and tell stories through my photographs.

</section>

<section class="contact">

<h2>Contact Me</h2>

<form>

<label for="name">Name:</label>

```
        <input type="text" id="name" name="name">
        <label for="email">Email:</label>
        <input type="email" id="email" name="email">
        <label for="message">Message:</label>
        <textarea id="message" name="message"></textarea>
        <button type="submit">Submit</button>
      </form>
    </section>
  </main>
  <footer>
    <p>&copy; 2023 Sammie Smith</p>
  </footer>
</body>
</html>
```

## CSS Code:

```
/* Global Styles */
* {
  box-sizing: border-box;
```

```
  margin: 0;
  padding: 0;
}
  body {
  font-family: Arial, sans-serif;
  font-size: 16px;
  line-height: 1.5;
  color: #130101;
}
  a {
  text-decoration: none;
  color: #4b0b0b;
}
  ul {
  list-style: none;
}
  img {
  max-width: 100%;
```

```
  height: auto;
}
  /* Header Styles */
  header {
  background-color: #7c7a7a;
  border-bottom: 1px solid #696262;
}
  nav ul {
  display: flex;
  justify-content: space-between;
 align-items: center;
  margin: 0;
  padding: 1rem;
}
  nav li {
  margin: 0 1rem;
}
  nav a {
```

```
  font-weight: bold;
  letter-spacing: 1px;
  text-transform: uppercase;
}
  /* Hero Section Styles */
  .hero {
  background-image: url(https://via.placeholder.com/1200x600);
  background-size: cover;
  background-position: center;
   height: 60vh;
   display: flex;
   flex-direction: column;
   justify-content: center;
  align-items: center;
  text-align: center;
}
  .hero h1 {
  font-size: 3rem;
```

```
  margin-bottom: 1rem;
  color: #791818;
  text-shadow: 2px 2px #333;
}
  .hero p {
  font-size: 1.5rem;
  color: #442222;
  text-shadow: 2px 2px #418d58;
}
  /* Portfolio Section Styles */
  .portfolio {
  padding: 2rem;
}
  .portfolio h2 {
  font-size: 2rem;
  margin-bottom: 1rem;
}
  .portfolio li {
```

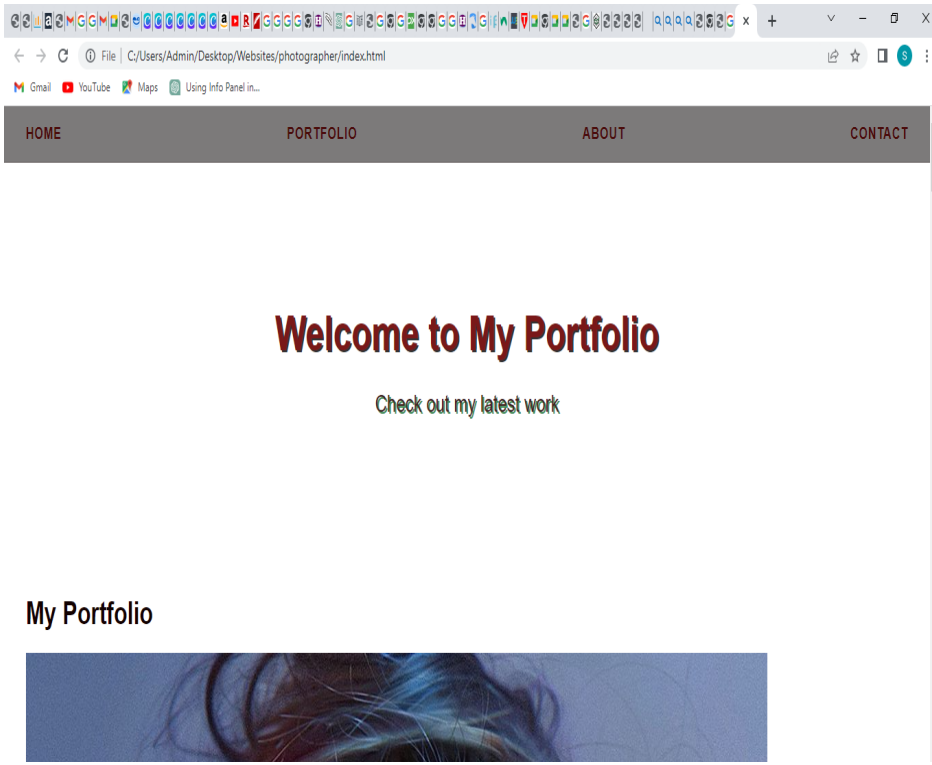
```
  margin-bottom: 2rem;
}
  .portfolio h3 {
  font-size: 1.5rem;
  margin-top: 1rem;
}
  /* About Section Styles */
  .about {
  padding: 2rem;
}
  .about h2 {
  font-size: 2rem;
  margin-bottom: 1rem;
}
  /* Contact Section Styles */
  .contact {
  padding: 2rem;
}
```

```
.contact h2 {
  font-size: 2rem;
  margin-bottom: 1rem;
}
label {
  display: block;
  margin-bottom: 0.5rem;
}
 input,
textarea {
  display: block;
  width: 100%;
  padding: 0.5rem;
  margin-bottom: 1rem;
  border: 1px solid #ccc;
  border-radius: 4px;
}
 button[type="submit"] {
```

```
  background-color: #333;
  color: #fff;
  border: none;
  padding: 0.5rem 1rem;
  border-radius: 4px;
  cursor: pointer;
  font-size: 1rem;
  text-transform: uppercase;
}
 button[type="submit"]:hover {
  background-color: #444;
}
 /* Footer Styles */
  footer {
  background-color: #333;
  color: #fff;
  text-align: center;
  padding: 1rem;
```

```
}
```

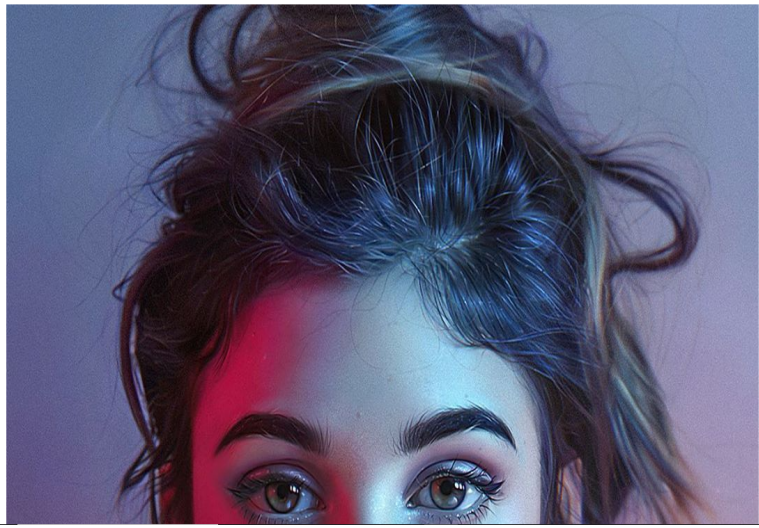
See the web result below.



File | C:/Users/Admin/Desktop/Websites/photographer/index.html

Gmail YouTube Maps Using Info Panel in...

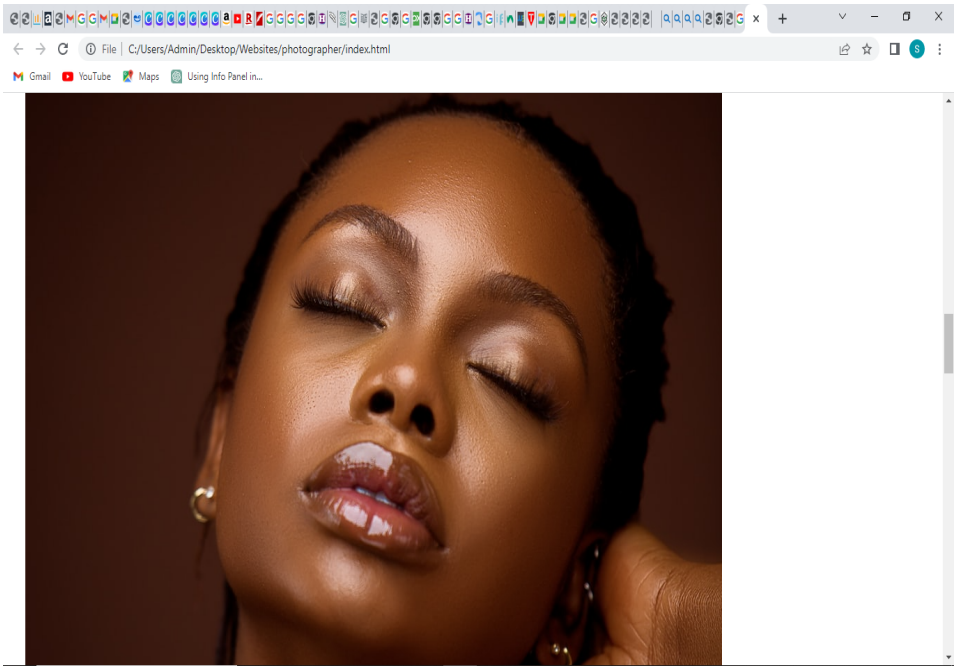
# My Portfolio



File | C:/Users/Admin/Desktop/Websites/photographer/index.html

Gmail YouTube Maps Using Info Panel in...

## Photograph 2




# Exercise 3: Develop a Responsive E-commerce Website

Develop a responsive e-commerce website with a homepage, product listing page, and product details page. Use responsive design techniques to ensure the website looks good on all devices and provides a seamless shopping experience with HTML and CSS.

Below is an example of a responsive e-commerce website.

**HTML Code:**

*<!DOCTYPE html>*

```
<html lang="en">

<head>

    <meta charset="UTF-8">

                <meta        http-equiv="X-UA-Compatible"
content="IE=edge">

       <meta   name="viewport"   content="width=device-
width, initial-scale=1.0">

    <title>Responsive E-commerce Website</title>

  <link rel="stylesheet" href="style.css">

</head>

<body>

  <header>

    <nav>

     <ul>

       <li><a href="#">Home</a></li>

       <li><a href="#">Shop</a></li>

       <li><a href="#">Contact</a></li>

       <li><a href="#">Cart</a></li>

     </ul>
```

```
    </nav>
  </header>
  <main>
    <section class="hero">
      <h1>Welcome to our E-commerce Website</h1>
      <p>Shop the latest products now</p>
      <a href="#" class="btn">Shop Now</a>
    </section>
    <section class="featured-products">
      <h2>Featured Products</h2>
      <ul>
        <li>
          <a href="#">
            <img src="img1.jpg" alt="Product 1">
            <h3>Product 1</h3>
            <p>$20.00</p>
          </a>
        </li>
```

```
      <li>
        <a href="#">
          <img src="img3.jpg" alt="Product 2">
          <h3>Product 2</h3>
          <p>$25.00</p>
        </a>
      </li>
      <li>
        <a href="#">
          <img src="img4.jpeg" alt="Product 3">
          <h3>Product 3</h3>
          <p>$30.00</p>
        </a>
      </li>
    </ul>
  </section>
</main>
<footer>
```

```
    <p>&copy; 2023 Sammie Smith</p>
  </footer>
</body>
</html>
```

## CSS Code:

```
/* Global Styles */
* {
    box-sizing: border-box;
    margin: 0;
    padding: 0;
 }
    body {
   font-family: Arial, sans-serif;
   font-size: 16px;
    line-height: 1.5;
 }
   a {
   text-decoration: none;
```

```
}

 ul {

  list-style: none;

}

 /* Header Styles */

header {

  background-color: #333;

  color: #d6d4d4;

  padding: 1rem;

}

  nav ul {

  display: flex;

  justify-content: space-between;

}

  nav ul li {

  margin-right: 1rem;

}

  nav ul li:last-child {
```

```
  margin-right: 0;
}
  nav ul li a {
  color: #fff;
}
  nav ul li a:hover {
  text-decoration: underline;
}
  /* Hero Styles */
.hero {
  background-image: url('https://via.placeholder.com/1920x1080');
  background-size: cover;
  background-position: center center;
  height: 500px;
  display: flex;
  flex-direction: column;
  justify-content: center;
  align-items: center;
```

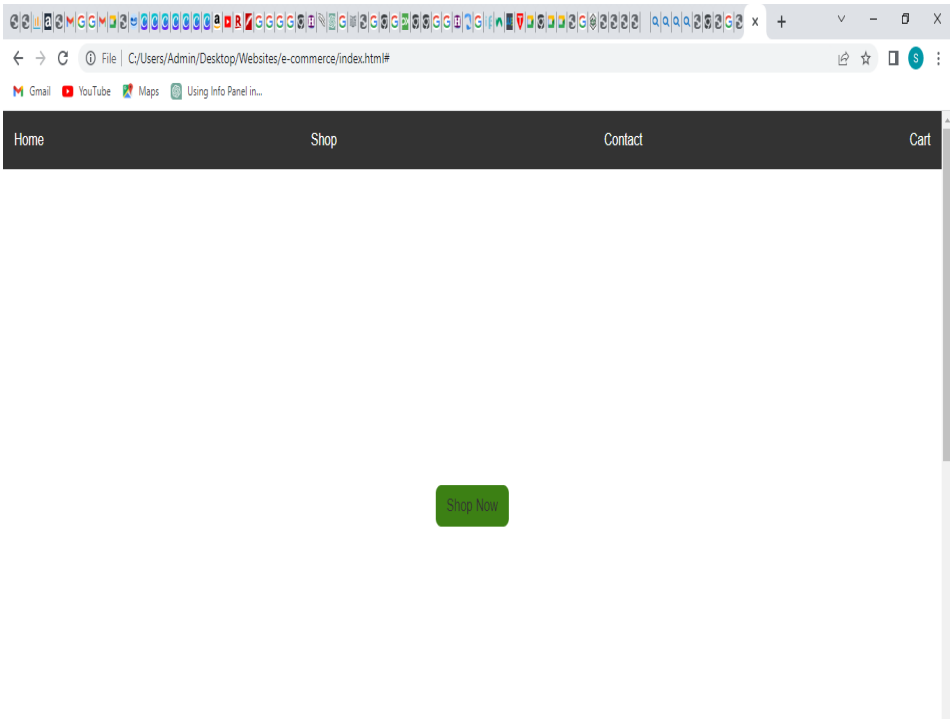
```
  text-align: center;
   color: #fff;
}
  .hero h1 {
  font-size: 3rem;
   margin-bottom: 1rem;
}
  .hero p {
  font-size: 1.5rem;
   margin-bottom: 1.5rem;
}
  .btn {
   background-color: #3c8014;
   color: #333;
   padding: 0.5rem 1rem;
   border-radius: 0.5rem;
  font-size: 1rem;
   transition: background-color 0.3s ease;
```

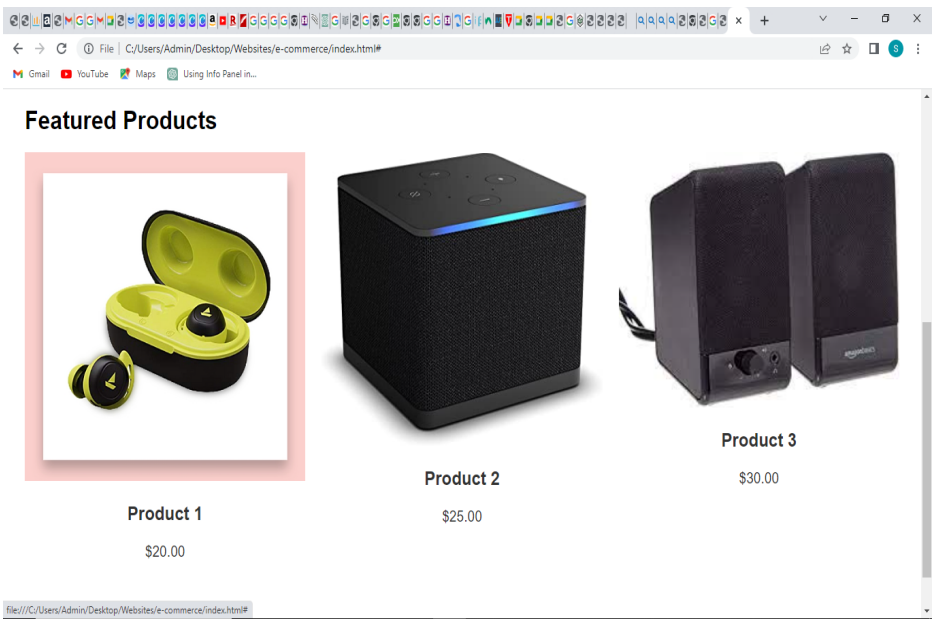
```
}

  .btn:hover {

    background-color: #1f725d;

    color: #f3f3f5;

   cursor: pointer;

}

   /* Featured Products Styles */

.featured-products {

    padding: 2rem;

}

   .featured-products h2 {

   font-size: 2rem;

    margin-bottom: 1rem;

}

   .featured-products ul {

    display: flex;

    justify-content: space-between;

   flex-wrap: wrap;
```

```
}

  .featured-products li {

    margin-bottom: 2rem;

  flex-basis: calc(33.33% - 1rem);

}

  .featured-products li a {

    display: block;

  text-align: center;

    color: #333;

}

  .featured-products li h3 {

    margin: 1rem 0;

  font-size: 1.5rem;

}

  .featured-products li p {

    margin-bottom: 1rem;

  font-size: 1.2rem;

}
```

```
  .featured-products li img {
    width: 100%;
    height: auto;
  }
  /* Footer Styles */
  footer {
    background-color: #adacac;
    padding: 1rem;
    text-align: center;
  }
```

See the results of the web view below.


File | C:/Users/Admin/Desktop/Websites/e-commerce/index.html#
Gmail
YouTube
Maps
Using Info Panel in...
Home
Shop
Contact
Cart
Shop Now


File | C:/Users/Admin/Desktop/Websites/e-commerce/index.html#
Gmail
YouTube
Maps
Using Info Panel in...
Featured Products
Product 1
$20.00
Product 2
$25.00
Product 3
$30.00
file:///C:/Users/Admin/Desktop/Websites/e-commerce/index.html#

## CONCLUSION

This book has offered a comprehensive guide to learning HTML and CSS, covering both the fundamental concepts and more advanced techniques. It has provided an overview of how to create accessible and search-engine-optimized web pages and has also given insights into the best practices and tips for working with HTML and CSS.

By using responsive design techniques, web developers can ensure that their websites look good on all devices, from desktop computers to smartphones and tablets. This is crucial in today's digital landscape, where users access websites from a variety of devices and screen sizes. Creating a responsive website with HTML and CSS requires

attention to detail and a solid understanding of both languages. Developers must be familiar with the latest HTML and CSS standards, as well as best practices for web design and development.

HTML and CSS are powerful tools for creating responsive websites that provide a seamless user experience across all devices. As technology continues to evolve, web developers must stay up-to-date with the latest trends and techniques to ensure that their websites remain relevant and effective in meeting the needs of their users.

With this knowledge, readers will be able to create visually appealing, responsive, and accessible web pages that meet the needs of their users. Whether you are a beginner or an experienced developer, this book has provided you with the tools and knowledge necessary to create websites that are both functional and visually appealing.

## INDEX

# A

alt, alt, alt, alt, alt, alt, alt, alt, alt, alt, alt, alt, alt, alt, alt, alt, alt, alt, alt, alt

alternative, alternative, alternative, alternative

animation, animation, animation, animation, animation, animation, animation, animation, animation, animation, animation, animation, animation, animation

Animations, Animations

animations, animations, animations, animations, animations, animations

Animations, Animations

animations

Animations, Animations, Animations, Animations, Animations

animations, animations, animations, animations

Animations

animations, animations

APIs

article, article, article, article, article

assistive, assistive, assistive

Atom, Atom

attribute, attribute, attribute,

audio, audio, audio, audio, audio, audio, audio, audio, audio, audio, audio, audio, audio, audio, audio

## B

blocks, blocks, blocks

body, body, body, body, body, body, body, body, body, body, body, body, body, body, body, body, body, body, body, body, body, body, body, body, body, body, body, body, body, body, body, body, body, body, body, body, body, body, body, body, body, body, body, body, body, body, body

border, border, border, border, border, border, border, border, border, border, border, border, border, border, border, border, border, border, border, border, border, border, border, border, border, border, border, border, border, border, border, border, border, border, border, border, border, border, border, border, border, border, border, border, border, border, border, border, border, border, border

borders, borders, borders, borders, borders, borders, borders

box model, box model, box model, box model, box model, box model, box model, box model, box model, box model

box-sizing, box-sizing, box-sizing, box-sizing, box-sizing

Brackets

browser, browser, browser, browser, browser, browser, browser, browser, browser, browser, browser, browser, browser, browser, browser, browser, browser, browser, browser, browser, browser, browser, browser, browser, browser, browser, browser, browser, browser, browser, browser, browser, browser, browser, browser, browser, browser, browser, browser, browser, browser, browser, browser, browser, browser, browser, browser, browser, browser, browser, browser, browser, browser, browser, browser

button, button, button, button, button, button, button, button, button, button, button, button, button, button, button, button, button, button, button, button, button, button, button, button, button, button, button, button, button, button, button, button, button, button

## C

Canvas, Canvas, Canvas, Canvas, Canvas, Canvas, Canvas, Canvas, Canvas, Canvas, Canvas

client, client, client, client, client

code, code, code,

color, color, color,

content, content, content, content, content, content, content, content, content,
content, content, content, content, content, content, content, content, content,
content, content, content, content, content, content, content, content, content,
content, content, content, content, content, content, content, content, content,
content, content, content, content, content, content, content, content, content,
content, content, content, content, content, content, content, content, content,
content, content, content, content, content, content, content, content, content,
content, content, content, content, content, content, content, content, content,
content, content, content, content, content, content, content, content, content,
content, content, content, content, content, content, content, content, content,
content, content, content, content, content, content, content, content, content,
content, content, content, content, content, content, content, content, content,

content, content, content, content, content, content, content, content, content

controls, controls, controls, controls, controls

create, create, create, create, create, create, create, create, create, create, create,
create, create, create, create, create, create, create, create, create, create, create,
create, create, create, create, create, create, create, create, create, create, create,
create, create, create, create, create, create, create, create, create, create, create,
create, create, create, create, create, create, create, create, create, create, create,
create, create, create, create, create, create, create, create, create, create, create,
create, create, create, create, create, create, create, create, create, create, create,
create, create, create, create, create, create, create, create, create, create, create,

create, create, create, create, create, create, create, create, create, create, create, create, create, create, create, create, create, create, create, create, create, create, create, create, create, create, create, create, create, create, create, create, create, create, create, create, create, create, create, create

Cross-platform

cryptographic

CSS, CSS, CSS, CSS, CSS, CSS, CSS, CSS, CSS, CSS, CSS, CSS, CSS, CSS,
CSS, CSS, CSS, CSS, CSS, CSS, CSS, CSS, CSS, CSS, CSS, CSS, CSS, CSS,
CSS, CSS, CSS, CSS, CSS, CSS, CSS, CSS, CSS, CSS, CSS, CSS, CSS, CSS,
CSS, CSS, CSS, CSS, CSS, CSS, CSS, CSS, CSS, CSS, CSS, CSS, CSS, CSS,
CSS, CSS, CSS, CSS, CSS, CSS, CSS, CSS, CSS, CSS, CSS, CSS, CSS, CSS,
CSS, CSS, CSS, CSS, CSS, CSS, CSS, CSS, CSS, CSS, CSS, CSS, CSS, CSS,
CSS, CSS, CSS, CSS, CSS, CSS, CSS, CSS, CSS, CSS, CSS, CSS, CSS, CSS,
CSS, CSS, CSS, CSS, CSS, CSS, CSS, CSS, CSS, CSS, CSS, CSS, CSS, CSS,
CSS, CSS, CSS, CSS, CSS, CSS, CSS, CSS, CSS, CSS, CSS, CSS, CSS, CSS,
CSS, CSS, CSS, CSS, CSS, CSS, CSS, CSS, CSS, CSS, CSS, CSS, CSS, CSS,
CSS, CSS, CSS, CSS, CSS, CSS, CSS, CSS, CSS, CSS, CSS, CSS, CSS, CSS,
CSS, CSS, CSS, CSS, CSS, CSS, CSS, CSS, CSS, CSS, CSS, CSS, CSS, CSS,
CSS, CSS, CSS, CSS, CSS, CSS, CSS, CSS, CSS, CSS, CSS, CSS, CSS, CSS,
CSS, CSS, CSS, CSS, CSS, CSS, CSS, CSS, CSS, CSS, CSS, CSS, CSS, CSS,

CSS, CSS, CSS,

display, display, display,

document, document, document, document, document, document, document, document, document, document, document, document, document, document, document

duration, duration, duration, duration, duration, duration

dynamic, dynamic, dynamic, dynamic, dynamic, dynamic, dynamic, dynamic, dynamic, dynamic, dynamic

## E

editor, editor, editor, editor, editor, editor, editor, editor, editor, editor, editor, editor, editor

elements, elements, elements,

features, features, features, features, features, features, features, features, features, features, features, features, features, features, features, features, features, features, features, features, features, features, features, features, features, features, features, features

Flexbox, Flexbox, Flexbox, Flexbox, Flexbox, Flexbox, Flexbox, Flexbox, Flexbox

fonts, fonts, fonts, fonts, fonts, fonts

footer, footer, footer, footer, footer, footer, footer, footer, footer, footer, footer, footer, footer, footer, footer, footer, footer, footer, footer, footer, footer, footer, footer, footer, footer, footer, footer, footer, footer

forms, forms, forms, forms, forms, forms, forms, forms, forms, forms

functional, functional, functional, functional, functional, functional

## H

head, head, head, head, head, head, head, head, head, head, head, head, head, head, head, head, head, head, head, head, head, head, head, head, head, head, head, head, head

header, header, header, header, header, header, header, header, header, header, header, header, header, header, header, header, header, header, header, header, header, header, header, header, header, header, header, header, header, header

heading, heading, heading, heading, heading, heading, heading

height, height, height, height, height, height, height, height, height, height, height, height, height, height, height, height, height, height, height, height, height, height, height, height, height, height, height, height, height, height

HTML, HTML, HTML, HTML, HTML, HTML, HTML, HTML, HTML,
HTML, HTML, HTML, HTML, HTML, HTML, HTML, HTML, HTML,
HTML, HTML, HTML, HTML, HTML, HTML, HTML, HTML, HTML,
HTML, HTML, HTML, HTML, HTML, HTML, HTML, HTML, HTML,
HTML, HTML, HTML, HTML, HTML, HTML, HTML, HTML, HTML,
HTML, HTML, HTML, HTML, HTML, HTML, HTML, HTML, HTML,
HTML, HTML, HTML, HTML, HTML, HTML, HTML, HTML, HTML,
HTML, HTML, HTML, HTML, HTML, HTML, HTML, HTML, HTML,

HTML, HTML, HTML,

I

ID, ID, ID, ID, ID, ID

IDE, IDE, IDE, IDE, IDE, IDE, IDE, IDE, IDE, IDE, IDE

images, images, images,

language, language, language, language, language, language

layout, layout, layout, layout, layout, layout, layout, layout, layout, layout, layout, layout, layout, layout, layout, layout, layout, layout, layout, layout, layout, layout, layout, layout, layout, layout, layout, layout, layout, layout, layout, layout, layout, layout

links

Links

links, links, links, links, links, links, links, links, links, links

Links

links, links

Links

links, links, links, links, links, links, links, links, links

list, list, list, list, list, list, list, list, list, list, list, list, list, list, list, list, list, list, list, list, list, list, list, list, list, list, list, list, list, list, list, list, list, list, list

lists, lists, lists, lists, lists, lists, lists, lists, lists, lists, lists, lists, lists, lists, lists, lists, lists, lists, lists, lists, lists, lists, lists, lists, lists

## M

Main

margin, margin, margin,

match, match

meta, meta, meta, meta, meta, meta, meta, meta, meta, meta, meta, meta, meta, meta, meta, meta, meta, meta, meta, meta, meta, meta, meta, meta, meta, meta, meta, meta, meta, meta, meta, meta, meta, meta, meta, meta, meta, meta, meta, meta, meta, meta, meta, meta, meta, meta, meta, meta, meta

Meter, Meter

minlength, minlength

modern, modern, modern, modern, modern, modern, modern, modern, modern, modern, modern, modern, modern, modern

multimedia, multimedia, multimedia, multimedia

multiple, multiple, multiple, multiple, multiple, multiple, multiple, multiple, multiple, multiple, multiple

## N

nav, nav, nav, nav, nav, nav, nav, nav, nav, nav, nav, nav, nav, nav, nav, nav, nav, nav, nav, nav, nav, nav, nav, nav, nav, nav, nav, nav, nav, nav, nav, nav, nav, nav, nav, nav, nav, nav, nav, nav, nav, nav, nav, nav, nav

navigation, navigation, navigation, navigation, navigation, navigation, navigation, navigation, navigation, navigation, navigation, navigation, navigation, navigation, navigation, navigation, navigation, navigation

Nested, Nested

## O

option, option, option, option, option, option, option

Ordered

ordered, ordered, ordered

Ordered, Ordered

ordered, ordered, ordered

Ordered

ordered, ordered, ordered

## P

padding, padding, padding,

pixels, pixels, pixels, pixels, pixels, pixels, pixels, pixels, pixels, pixels, pixels, pixels, pixels, pixels

Placeholders

Preload

presentation, presentation, presentation, presentation, presentation, presentation, presentation, presentation, presentation

professional, professional, professional

Progress, Progress

properties, properties, properties,

property, property, property,

Responsive, Responsive, Responsive, Responsive, Responsive

responsive

Responsive, Responsive

responsive

Responsive

responsive, responsive, responsive, responsive, responsive, responsive,
responsive, responsive, responsive, responsive, responsive, responsive,
responsive, responsive, responsive, responsive, responsive

Responsive

responsive, responsive, responsive

Responsive

responsive

Responsive

responsive

Responsive, Responsive

responsive

Responsive

responsive

Responsive, Responsive

responsive, responsive

Responsive, Responsive

responsive, responsive, responsive

Responsive

responsive, responsive, responsive, responsive, responsive

Responsive, Responsive

responsive, responsive, responsive

Responsive, Responsive

responsive, responsive, responsive

Responsive

responsive, responsive, responsive, responsive, responsive, responsive, responsive

Responsive

responsive, responsive

Responsive

responsive, responsive

Responsive

responsive, responsive, responsive

Reverse

rows, rows, rows, rows, rows, rows, rows, rows, rows, rows, rows, rows, rows, rows, rows

## S

Scalable, Scalable, Scalable, Scalable, Scalable

screen, screen, screen,

semantic, semantic, semantic, semantic, semantic, semantic, semantic, semantic, semantic

server, server, server, server, server, server, server, server, server

sizes, sizes, sizes, sizes, sizes, sizes, sizes, sizes, sizes, sizes, sizes, sizes, sizes, sizes, sizes, sizes, sizes, sizes, sizes, sizes, sizes, sizes, sizes, sizes, sizes, sizes, sizes, sizes, sizes, sizes, sizes, sizes, sizes

skill, skill, skill, skill, skill

skills, skills, skills, skills, skills, skills

smartphones, smartphones, smartphones, smartphones, smartphones

Sockets, Sockets, Sockets, Sockets

style, style, style, style, style, style, style, style, style, style, style, style, style, style, style, style, style, style, style, style, style, style, style, style, style, style, style, style, style, style, style, style, style, style, style, style, style, style, style, style, style, style, style, style, style, style, style, style, style,

style, style, style, style

Styling, Styling, Styling

subheading

Sublime, Sublime, Sublime, Sublime

submit, submit, submit, submit, submit, submit, submit, submit, submit, submit, submit, submit, submit

SVG, SVG, SVG, SVG, SVG, SVG, SVG, SVG, SVG, SVG, SVG, SVG, SVG, SVG, SVG, SVG, SVG, SVG, SVG, SVG

## T

Tables, Tables, Tables, Tables, Tables, Tables

tablets, tablets, tablets, tablets, tablets, tablets

tag, tag, tag,

trends, trends

type, type, type,

user, user, user,

value, value, value, value, value

video, video, video, video, video, video, video, video, video, video, video, video, video, video, video

Videos, Videos

videos, videos, videos

Videos

videos, videos, videos

Visual Studio, Visual Studio, Visual Studio, Visual Studio, Visual Studio

## W

Web, Web, Web, Web, Web, Web, Web

web, web, web, web, web

Web

web, web, web, web, web, web

Web

web, web, web, web, web, web, web

Web

web, web, web,

web, web, web, web, web, web, web, web, web, web, web, web, web, web, web, web, web, web, web, web, web, web, web, web, web, web, web, web, web, web, web, web, web, web, web, web, web, web, web, web

Web

web, web, web, web, web, web

Web

web, web, web, web, web, web, web, web, web, web, web, web, web, web

Web

web, web, web, web, web, web, web, web, web

web pages, web pages, web pages, web pages, web pages

website, website, website, website, website, website, website, website, website, website, website, website, website, website, website, website, website, website, website, website, website, website, website, website, website, website, website, website, website, website, website, website, website, website, website, website, website, website, website, website, website, website, website, website, website,

website, website, website, website, website, website, website, website, website, website, website, website, website, website, website, website, website, website, website, website, website, website, website, website, website, website, website

websites, websites, websites, websites, websites, websites, websites, websites, websites, websites, websites, websites, websites, websites, websites, websites, websites, websites, websites, websites, websites, websites, websites, websites, websites, websites, websites, websites, websites, websites, websites, websites, websites, websites, websites, websites, websites, websites, websites, websites, websites, websites, websites, websites, websites

WebSocket, WebSocket, WebSocket

WebStorm, WebStorm

width, width, width, width, width, width, width, width, width, width, width, width, width, width, width, width, width, width, width, width, width, width, width, width, width, width, width, width, width, width, width, width, width, width, width, width, width, width, width, width, width, width, width, width, width, width, width, width, width, width, width, width, width, width, width, width, width, width, width, width, width, width, width